PointNet, PointNet++ Hand Pose Estimation

2019.7.17 By Meiyu Wan

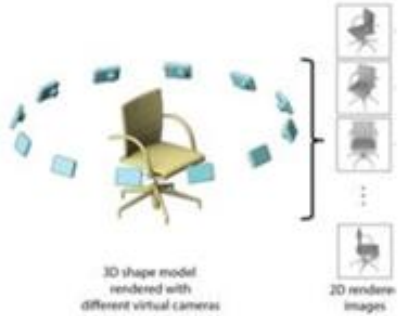
index

PointNet

- 多视角（multi-view）：通过多视角二维图片组合为三维物体，此方法将传统CNN应用于多张二维视角的图片，特征被view pooling procedure聚合起来形成三维物体；
- 体素（volumetric）：通过将物体表现为空间中的体素进行类似于二维的三维卷积（例如，卷积核大小为5x5x5），是规律化的并且易于类比二维的，但同时因为多了一个维度出来，时间和空间复杂度都非常高，目前已经不是主流的方法了；
- 点云（point clouds）：直接将三维点云抛入网络进行训练，数据量小。主要任务有分类、分割以及大场景下语义分割；
- 非欧式（manifold，graph）：在流形或图的结构上进行卷积，三维点云可以表现为mesh结构，可以通过点对之间临接关系表现为图的结构。

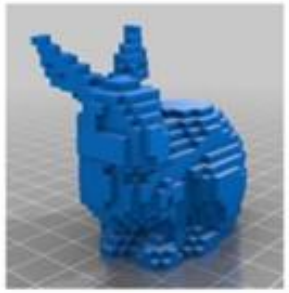
Cont’d

- Projec



[Su et al. 2015]
[Kalogerakis et al. 2016]
…

Multi-view

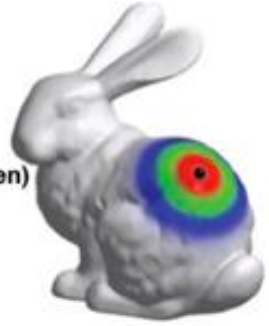


[Maturana et al. 2015]
[Wu et al. 2015] (GAN)
[Qi et al. 2016]
[Liu et al. 2016]
[Wang et al. 2017] (O-Net)
[Tatarchenko et al. 2017] (OGN)
…

Volumetric

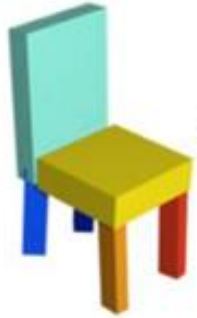
[Qi et al. 2017] (PointNet)
[Fan et al. 2017] (PointSetGen)

Point cloud



[Defferard et al. 2016]
[Henaff et al. 2015]
[Yi et al. 2017] (SyncSpecCNN)
…

Mesh (Graph CNN)

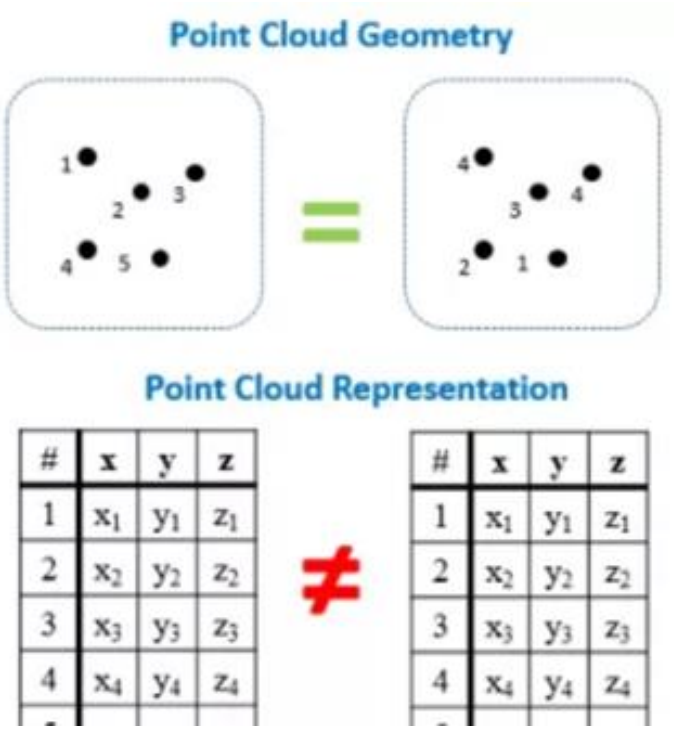


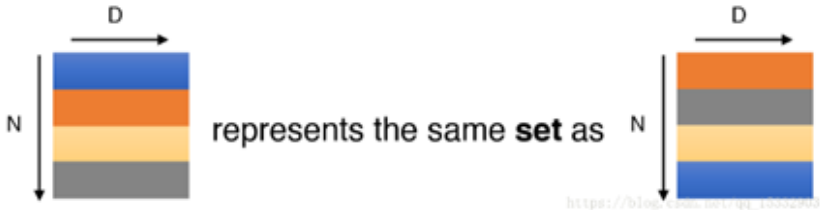
[Tulsiani et al. 2017]
[Li et al. 2017] (GRASS)

Part assembly

Challenge of Point Cloud

- 无序性：点云本质上是一长串点（nx3矩阵，其中n是点数）。在几何上，点的顺序不影响它在空间中对整体形状的表示，例如，相同的点云可以由两个完全不同的矩阵表示。如右图上边所示：
- 我们希望得到的效果如右图下边：N代表点云个数，D代表每个点的特征维度。不论点云顺序怎样，希望得到相同的特征提取结果。

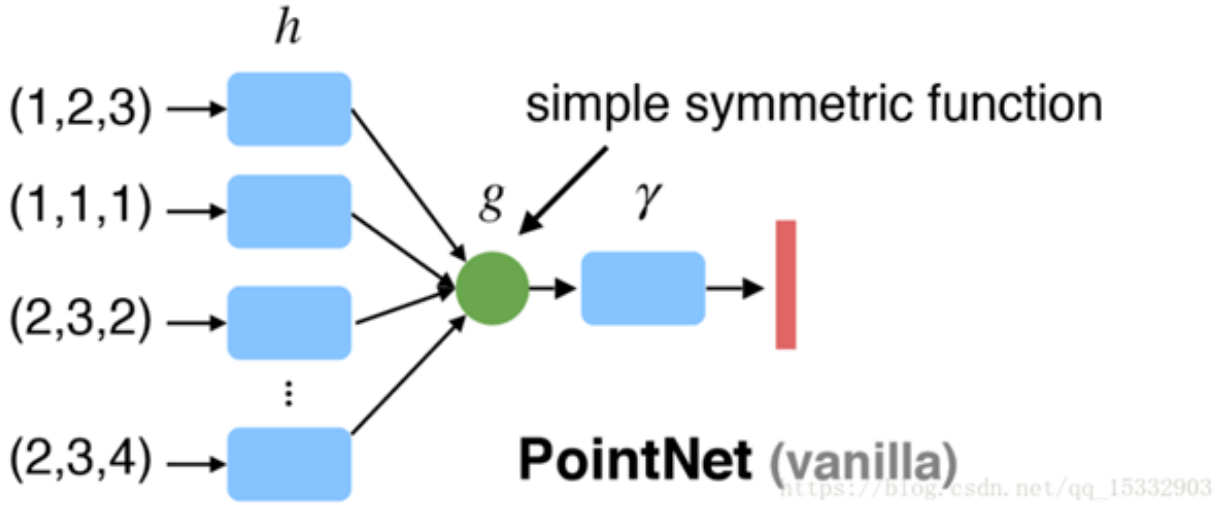




无序性

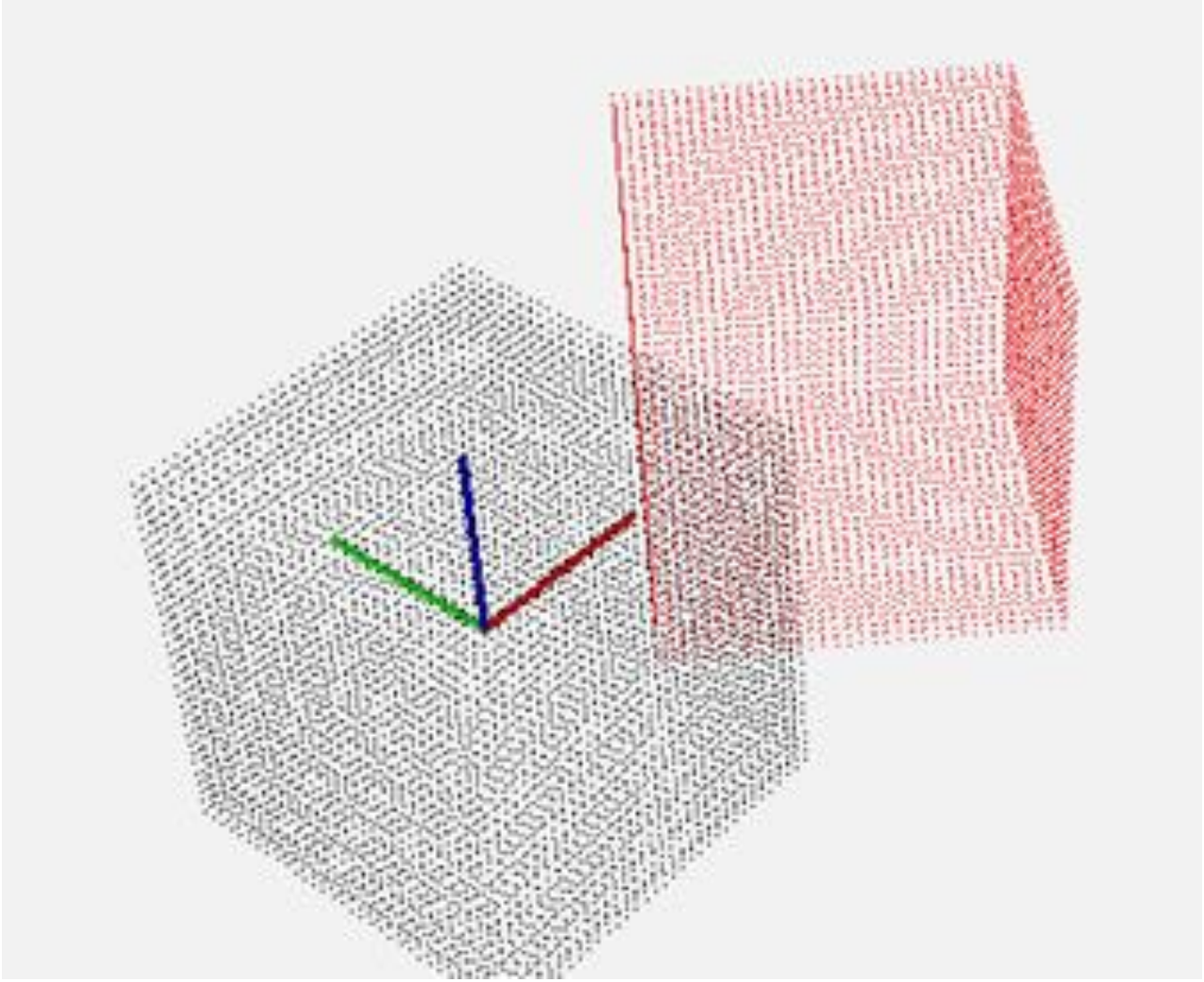
- 我们知道，网络的一般结构是：提特征-特征映射-特征图压缩（降维）-全连接。
- 右图中x代表点云中某个点，h代表特征提取层，g叫做对称方法，r代表更高维特征提取，最后接一个softmax分类。g可以是maxpooling或sumpooling，也就是说，最后的D维特征对每一维都选取N个点中对应的最大特征值或特征值总和，这样就可以通过g来解决无序性问题。pointnet采用了max-pooling策略。

$$f(x_1,x_2,\ldots,x_n) = \gamma \circ g(h(x_1),\ldots,h(x_n))$$



旋转性

- 相同的点云在空间中经过一定的刚性变化（旋转或平移），坐标发生变化。我们希望不论点云在怎样的坐标系下呈现，网络都能正确的识别出。这个问题可以通过STN（spacial transform netw）来解决。三维不是点云是一个不规则的结构（无序，无网格）。pointnet通过学习一个矩阵来达到对目标最有效的变换。



点与点之间的空间关系

- 一个物体通常由特定空间内的一定数量的点云构成，也就是说这些点云之间存在空间关系。为了能有效利用这种空间关系，论文作者提出了将局部特征和全局特征进行串联的方式来聚合信息。

Two principles

Theorem 1. *Suppose* $f : \mathcal{X} \to \mathbb{R}$ *is a continuous set function w.r.t Hausdorff distance* $d_H(\cdot,\cdot)$. $\forall \epsilon > 0$, $\exists$ *a continuous function* h *and a symmetric function* $g(x_1,\ldots,x_n) = \gamma \circ MAX$, *such that for any* $S \in \mathcal{X}$,

$$\left| f(S) - \gamma\left(\underset{x_i \in S}{MAX}\{h(x_i)\}\right)\right| < \epsilon$$

- 定理1证明了PointNet的网络结构能够拟合任意的连续集合函数。其作用类似证明神经网络能够拟合任意连续函数一样。同时，作者发现PointNet模型的表征能力和maxpooling操作输出的数据维度(K)相关，K值越大，模型的表征能力越强。

Two principles

Theorem 2. *Suppose* $\mathbf{u} : \mathcal{X} \to \mathbb{R}^K$ *such that* $\mathbf{u} = \underset{x_i \in S}{MAX}\{h(x_i)\}$ *and* $f = \gamma \circ \mathbf{u}$. *Then,*

(a) $\forall S, \exists\, \mathcal{C}_S, \mathcal{N}_S \subseteq \mathcal{X}$, $f(T) = f(S)$ *if* $\mathcal{C}_S \subseteq T \subseteq \mathcal{N}_S$;

(b) $|\mathcal{C}_S| \leq K$

- 定理2(a)说明对于任何输入数据集S，都存在一个最小集Cs和一个最大集Ns，使得对Cs和Ns之间的任何集合T，其网络输出都和S一样。这也就是说，模型对输入数据在有噪声(引入额外的数据点，趋于Ns)和有数据损坏(缺少数据点，趋于Cs)的情况都是鲁棒的。定理2(b)说明了最小集Cs的数据多少由maxpooling操作输出数据的维度K给出上界。换个角度来讲，PointNet能够总结出表示某类物体形状的关键点，基于这些关键点PointNet能够判别物体的类别。这样的能力决定了PointNet对噪声和数据缺失的鲁棒性。

Cont’d

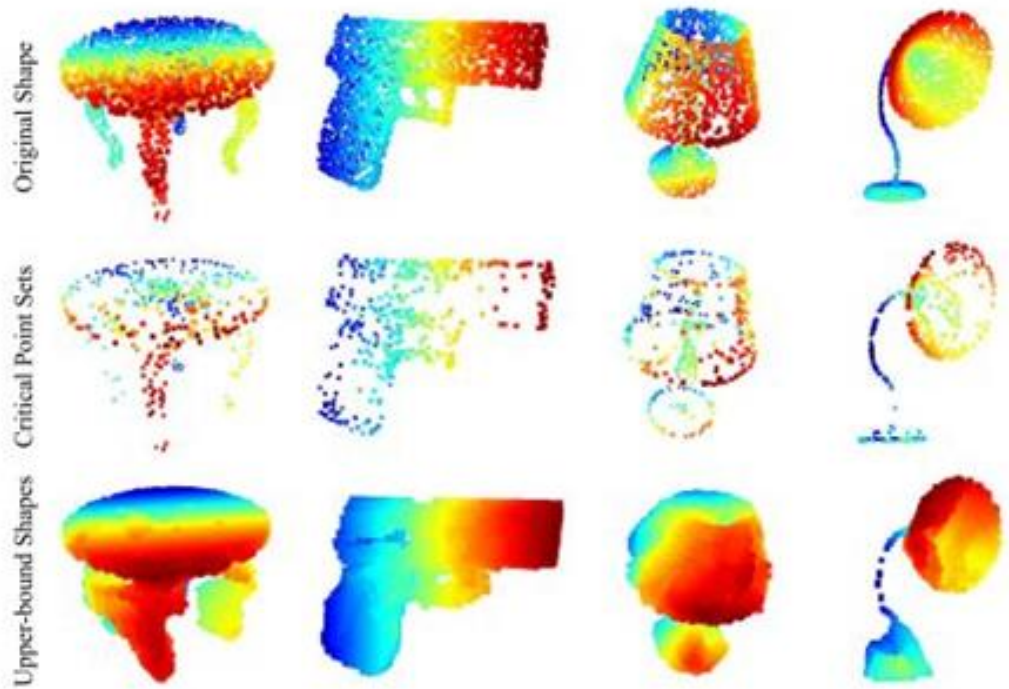


Figure 7. **Critical points and upper bound shape.** While critical points jointly determine the global shape feature for a given shape, any point cloud that falls between the critical points set and the upper bound shape gives exactly the same feature. We color-code all figures to show the depth information.

PointNet

- 先来看网络的两个亮点：
- 1.空间变换网络解决旋转问题：三维的STN可以通过学习点云本身的位姿信息学习到一个最有利于网络进行分类或分割的DxD旋转矩阵（D代表特征维度，pointnet中D采用3和64）。其中的原理是，通过控制最后的loss来对变换矩阵进行调整，pointnet并不关心最后真正做了什么变换，只要有利于最后的结果都可以。pointnet采用了两次STN，第一次input transform是对空间中点云进行调整，直观上理解是旋转出一个更有利于分类或分割的角度，比如把物体转到正面；第二次feature transform是对提取出的64维特征进行对齐，即在特征层面对点云进行变换。
- 2.maxpooling解决无序性问题：网络对每个点进行了一定程度的特征提取之后，maxpooling可以对点云的整体提取出global feature。

Cont'd

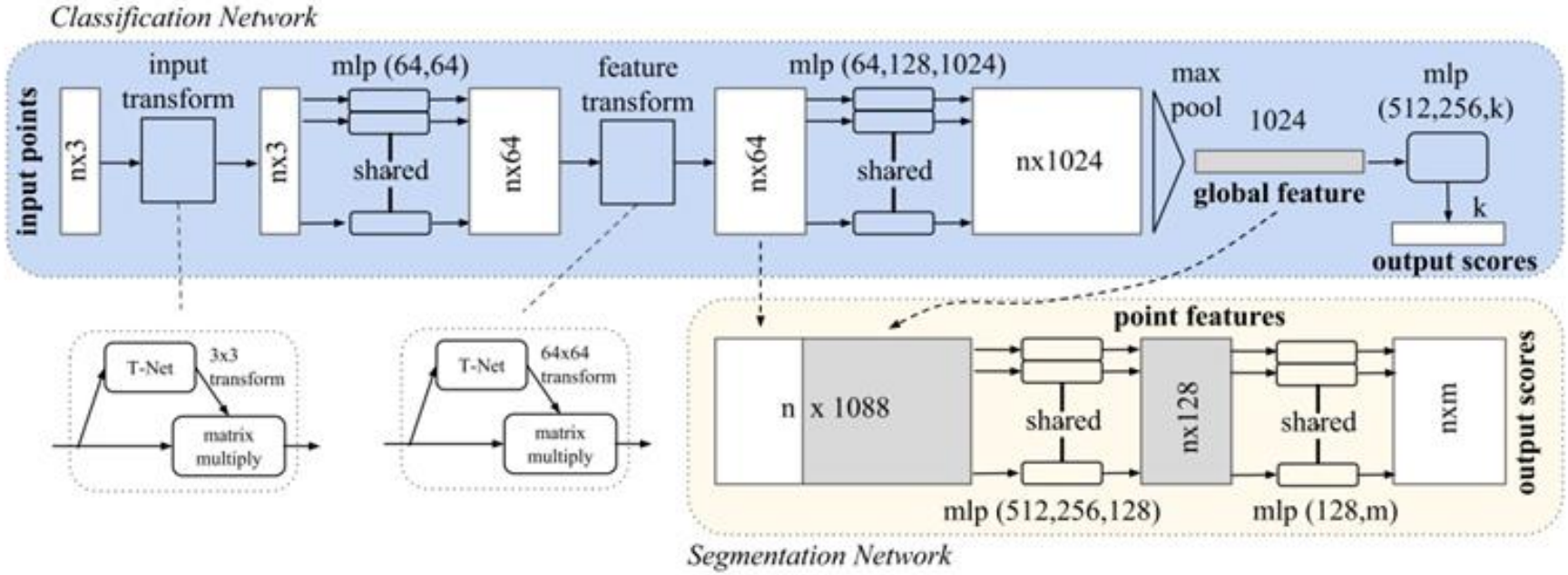


Figure 2. **PointNet Architecture.** The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. "mlp" stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net.

流程

- 1、输入为一帧的全部点云数据的集合，表示为一个nx3的2d tensor，其中n代表点云数量，3对应xyz坐标。
- 2、输入数据先通过和一个T-Net学习到的转换矩阵相乘来对齐，保证了模型的对特定空间转换的不变性。
- 3、通过多次mlp对各点云数据进行特征提取后，再用一个T-Net对特征进行对齐。
- 4、在特征的各个维度上执行maxpooling操作来得到最终的全局特征。
- 5、对分类任务，将全局特征通过mlp来预测最后的分类分数；对分割任务，将全局特征和之前学习到的各点云的局部特征进行串联，再通过mlp得到每个数据点的分类结果。

code

```
with tf.variable_scope('transform_net2') as sc:
    transform = feature_transform_net(net, is_training, bn_decay, K=64)
end_points['transform'] = transform
net_transformed = tf.matmul(tf.squeeze(net, axis=[2]), transform)
net_transformed = tf.expand_dims(net_transformed, [2])

net = tf_util.conv2d(net_transformed, 64, [1,1],
                     padding='VALID', stride=[1,1],
                     bn=True, is_training=is_training,
                     scope='conv3', bn_decay=bn_decay)
net = tf_util.conv2d(net, 128, [1,1],
                     padding='VALID', stride=[1,1],
                     bn=True, is_training=is_training,
                     scope='conv4', bn_decay=bn_decay)
net = tf_util.conv2d(net, 1024, [1,1],
                     padding='VALID', stride=[1,1],
                     bn=True, is_training=is_training,
                     scope='conv5', bn_decay=bn_decay)
```

```
def get_model(point_cloud, is_training, bn_decay=None):
    """ Classification PointNet, input is BxNx3, output Bx40 """
    batch_size = point_cloud.get_shape()[0].value
    num_point = point_cloud.get_shape()[1].value
    end_points = {}

    with tf.variable_scope('transform_net1') as sc:
        transform = input_transform_net(point_cloud, is_training, bn_decay, K=3)
    point_cloud_transformed = tf.matmul(point_cloud, transform)
    input_image = tf.expand_dims(point_cloud_transformed, -1)

    net = tf_util.conv2d(input_image, 64, [1,3],
                         padding='VALID', stride=[1,1],
                         bn=True, is_training=is_training,
                         scope='conv1', bn_decay=bn_decay)
    net = tf_util.conv2d(net, 64, [1,1],
                         padding='VALID', stride=[1,1],
                         bn=True, is_training=is_training,
                         scope='conv2', bn_decay=bn_decay)
```

Cont'd

```
# Symmetric function: max pooling
net = tf_util.max_pool2d(net, [num_point,1],
                         padding='VALID', scope='maxpool')

net = tf.reshape(net, [batch_size, -1])
net = tf_util.fully_connected(net, 512, bn=True, is_training=is_training,
                              scope='fc1', bn_decay=bn_decay)
net = tf_util.dropout(net, keep_prob=0.7, is_training=is_training,
                      scope='dp1')
net = tf_util.fully_connected(net, 256, bn=True, is_training=is_training,
                              scope='fc2', bn_decay=bn_decay)
net = tf_util.dropout(net, keep_prob=0.7, is_training=is_training,
                      scope='dp2')
net = tf_util.fully_connected(net, 40, activation_fn=None, scope='fc3')

return net, end_points
```

STN

```
def input_transform_net(point_cloud, is_training, bn_decay=None, K=3):
    """ Input (XYZ) Transform Net, input is BxNx3 gray image
        Return:
            Transformation matrix of size 3xK """
    batch_size = point_cloud.get_shape()[0].value
    num_point = point_cloud.get_shape()[1].value

    input_image = tf.expand_dims(point_cloud, -1)
    net = tf_util.conv2d(input_image, 64, [1,3],
                         padding='VALID', stride=[1,1],
                         bn=True, is_training=is_training,
                         scope='tconv1', bn_decay=bn_decay)
    net = tf_util.conv2d(net, 128, [1,1],
                         padding='VALID', stride=[1,1],
                         bn=True, is_training=is_training,
                         scope='tconv2', bn_decay=bn_decay)
    net = tf_util.conv2d(net, 1024, [1,1],
                         padding='VALID', stride=[1,1],
                         bn=True, is_training=is_training,
                         scope='tconv3', bn_decay=bn_decay)
    net = tf_util.max_pool2d(net, [num_point,1],
                             padding='VALID', scope='tmaxpool')

    net = tf.reshape(net, [batch_size, -1])
    net = tf_util.fully_connected(net, 512, bn=True, is_training=is_training,
                                  scope='tfc1', bn_decay=bn_decay)
    net = tf_util.fully_connected(net, 256, bn=True, is_training=is_training,
                                  scope='tfc2', bn_decay=bn_decay)
```

PointNet++

- PointNet提取特征的方式是对所有点云数据提取了一个全局的特征，显然，这和目前流行的CNN逐层提取局部特征的方式不一样。受到CNN的启发，作者提出了PointNet++，它能够在不同尺度提取局部特征，通过多层网络结构得到深层特征。

Cont’d

PointNet++由以下几个关键部分构成：

- 采样层（sampling）
- 激光雷达单帧的数据点可以多达100k个，如果对每一个点都提取局部特征，计算量是非常巨大的。因此，作者提出了先对数据点进行采样。作者使用的采样算法是最远点采样（farthest point sampling, FPS），相对于随机采样，这种采样算法能够更好地覆盖整个采样空间。
- 组合层（grouping）
- 为了提取一个点的局部特征，首先需要定义这个点的“局部”是什么。一个图片像素点的局部是其周围一定曼哈顿距离下的像素点，通常由卷积层的卷积核大小确定。同理，点云数据中的一个点的局部由其周围给定半径划出的球形空间内的其他点构成。组合层的作用就是找出通过采样层后的每一个点的所有构成其局部的点，以方便后续对每个局部提取特征。
- 特征提取层（feature learning）
- 因为PointNet给出了一个基于点云数据的特征提取网络，因此可以用PointNet对组合层给出的各个局部进行特征提取来得到局部特征。值得注意的是，虽然组合层给出的各个局部可能由不同数量的点构成，但是通过PointNet后都能得到维度一致的特征（由上述K值决定）。

cont’d

- 上述各层构成了PointNet++的基础处理模块。如果将多个这样的处理模块级联组合起来，PointNet++就能像CNN一样从浅层特征得到深层语义特征。对于分割任务的网络，还需要将下采样后的特征进行上采样，使得原始点云中的每个点都有对应的特征。这个上采样的过程通过最近的k个临近点进行插值计算得到。完整的PointNet++的网络示意图如下图所示。

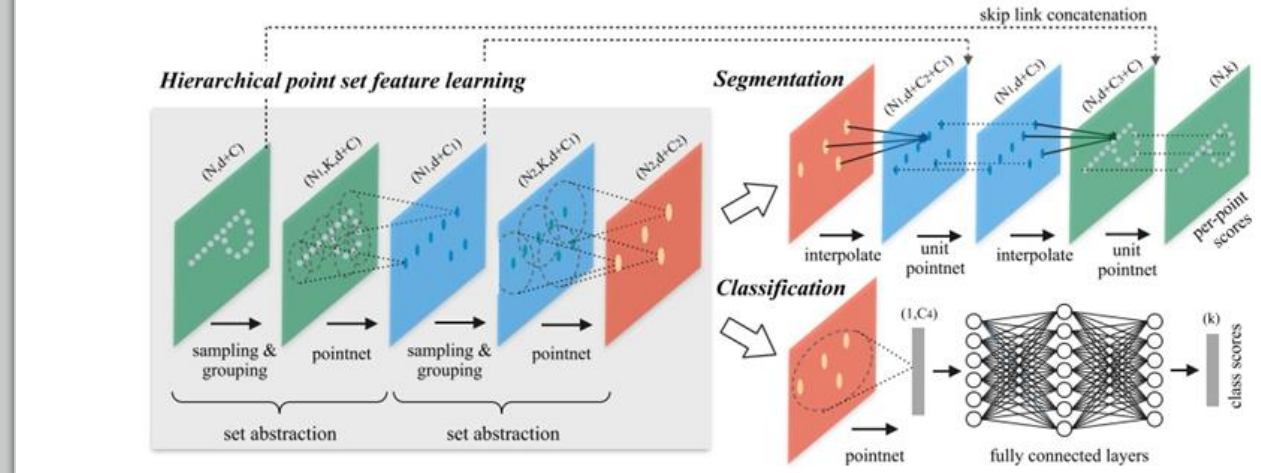


Figure 2: Illustration of our hierarchical feature learning architecture and its application for set segmentation and classification using points in 2D Euclidean space as an example. Single scale point grouping is visualized here. For details on density adaptive grouping, see Fig. 3

不均匀点云数据的特征提取

- 不同于图片数据分布在规则的像素网格上且有均匀的数据密度，点云数据在空间中的分布是不规则且不均匀的。虽然PointNet能够用于对各个点云局部提取特征，但是由于点云在各个局部均匀性不一致，很可能导致学习到的PointNet不能提取到很好的局部特征。比如说，在越远的地方激光雷达数据通常变得越稀疏，因此在稀疏的地方应该考虑更大的尺度范围来提取特征。为此，作者提出了两种组合策略来保证更优的特征提取。
- 如前所述，点集通常在不同区域具有不均匀的密度。这种不均匀性为点集特征学习带来了重大挑战。在密集数据中学习的特征可能不会推广到稀疏采样区域。因此，针对稀疏点云训练的模型可能无法识别细粒度的局部结构。

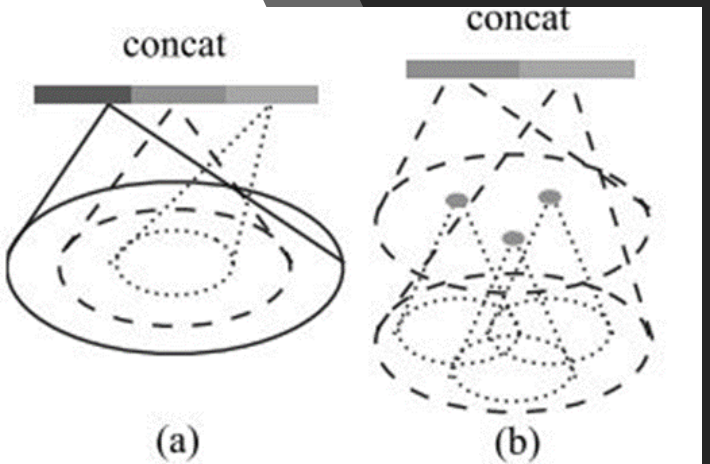


Figure 3: (a) Multi-scale grouping (MSG); (b) Multi-resolution grouping (MRG).

- 多尺度组合（multi-scale grouping, MSG）：
- 比较直接的想法是对不同尺度的局部提取特征并将它们串联在一起，如下图(a)所示。但是因为需要对每个局部的每个尺度提取特征，其计算量的增加也是很显著的。
- 【我们对网络进行培训，以学习优化策略，以结合多尺度功能。 这是通过随机丢弃每个实例的随机概率输入点来完成的，我们将其称为随机输入丢失。具体地，对于每个训练点集，我们选择从[0，p]均匀采样的丢失比θ，其中p≤1。对于每个点，我们以概率θ随机地丢弃一个点。 在实践中，我们设置p = 0.95以避免生成空点集。 在这样做时，我们向网络呈现各种稀疏度（由θ引起）和变化均匀性（由丢失中的随机性引起）的训练集。 在测试期间，我们保留所有可用点。】
- 多分辨率组合（multi-resolution grouping, MRG）：
- 为了解决MSG计算量太大的问题，作者提出了MRG。此种方法在某一层对每个局部提取到的特征由两个向量串联构成，如下图(b)所示。第一部分由其前一层提取到的特征再次通过特征提取网络得到，第二部分则通过直接对这个局部对应的原始点云数据中的所有点进行特征提取得到。

result

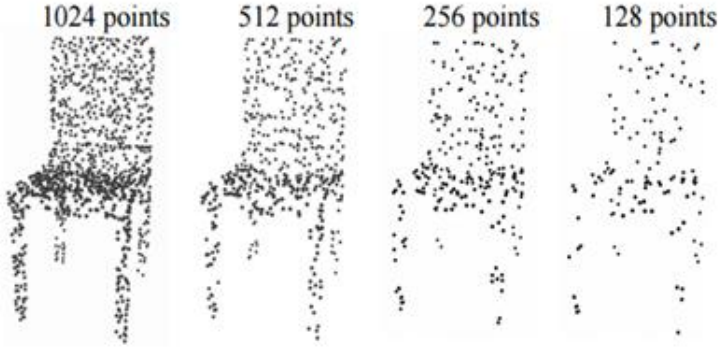
左：具有随机点丢失的点云。 右图：曲线显示我们的密度自适应策略在处理非均匀密度方面的优势。 DP表示训练期间的随机输入丢失; 否则训练是在均匀密集点上进行的。

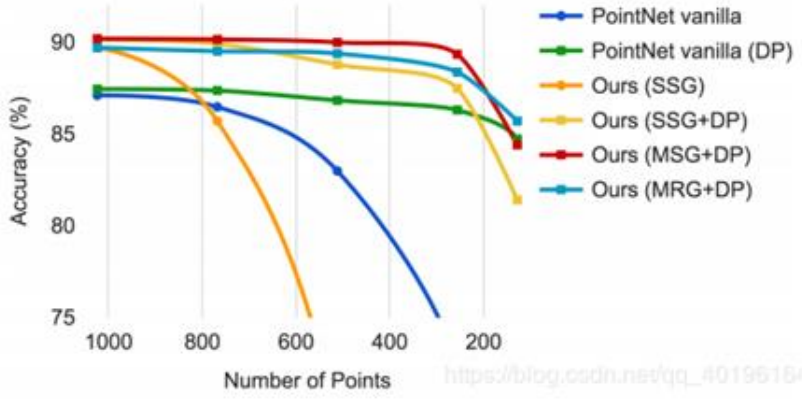
Method	Error rate (%)
Multi-layer perceptron [24]	1.60
LeNet5 [11]	0.80
Network in Network [13]	**0.47**
PointNet (vanilla) [20]	1.30
PointNet [20]	0.78
Ours	0.51

Table 1: MNIST digit classification.

Method	Input	Accuracy (%)
Subvolume [21]	vox	89.2
MVCNN [26]	img	90.1
PointNet (vanilla) [20]	pc	87.2
PointNet [20]	pc	89.2
Ours	pc	90.7
Ours (with normal)	pc	**91.9**

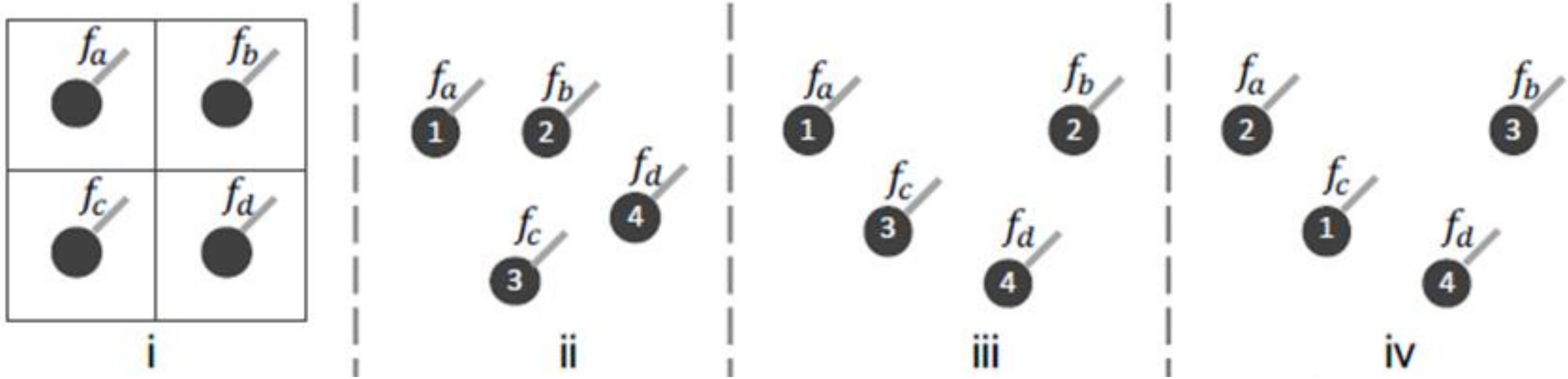
Table 2: ModelNet40 shape classification.





PointCNN

这篇论文先举例子解释了为什么卷积无法直接应用在点云数据上。



如图1， 传统的卷积是作用在2维图像数据上。图像中每个像素的顺序是固定的，也就是说数据是结构化存储的。直接使用conv2d就能从这种潜在的空间结构中获取信息。
而点云数据是点集，如果直接使用卷积会出现图中234多种情况

- 若直接使用卷积，则f2与f3的计算结果是相等的，但是从图中可知，23显示不同，这说明卷积无法获得点的空间信息。 而f3与f4的计算结果不等，但是图3与图4是相同的点集，必须得到相同的计算结果才合理，这说明卷积无法适应点集的N！种排列。在其他论文里，为了适应点云数据的这两种的特点采取的方式有体素化、3DCNN及PointNet提的对称操作。而PointCNN里采用的是这样的策略：
- 类似于空间变换网络（STN），从前一层的数据中取K个点，预测一个KxK大小的变换矩阵（X-transformation），用X矩阵前一层的特征做变换，然后对变换后的特征用卷积。

$$f_{ii} = Conv(\mathbf{K}, [f_a, f_b, f_c, f_d]^T),$$
$$f_{iii} = Conv(\mathbf{K}, [f_a, f_b, f_c, f_d]^T),$$
$$f_{iv} = Conv(\mathbf{K}, [f_c, f_a, f_b, f_d]^T),$$

$$f_{ii} = Conv(\mathbf{K}, \mathcal{X}_{ii} \times [f_a, f_b, f_c, f_d]^T),$$
$$f_{iii} = Conv(\mathbf{K}, \mathcal{X}_{iii} \times [f_a, f_b, f_c, f_d]^T),$$
$$f_{iv} = Conv(\mathbf{K}, \mathcal{X}_{iv} \times [f_c, f_a, f_b, f_d]^T),$$

Hierarchical Convolution 分层卷积

- CNN的输入是[R1,R1,C1]而卷积核的大小是[K,K,C1,C2]从前一层F1中大小为[K, K, C1]的区域生成F2中形状大小为[R2, R2, C2]的特征。其中R2<R1, C2>C1，即特征图的分辨率降低，但特征图数量增加，得到较高层次的信息。
- 　而PointCNN的输入时点集F1 = {(p1,i , f1,i) : i = 1, 2, ..., N1},， 其中p是点的D维坐标（作者没有假设点的维数，所以这里用D，如果是3维空间D就是3），f是点对应的特征。
- 对F1用 X-Conv（就是带X变换的卷积）得到F2 = {(p2,i , f2,i) : f2,i ∈ RC2 , i =1, 2, ..., N2}。仿照CNN，这里也要求N2<N1, C2>C1， 所以随着一层一层的映射，点的数量会越来越少，而每个点的特征会越来越多

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

Input : K, p, P, F

Output: F_p ▷ Features "projected", or "aggregated", to p

1: $P' \leftarrow P - p$ ▷ Move P to local coordinate system of p

2: $F_\delta \leftarrow MLP_\delta(P')$ ▷ **Individually** lift each point into C_δ dim. space

3: $F_* \leftarrow [F_\delta, F]$ ▷ Concatenate F_δ and F, F_* is a $K \times (C_\delta + C_1)$ matrix

4: $\mathcal{X} \leftarrow MLP(P')$ ▷ Learn the $K \times K$ $\mathcal{X}$-transformation matrix

5: $F_\mathcal{X} \leftarrow \mathcal{X} \times F_*$ ▷ Weight and permute F_* with the learnt $\mathcal{X}$

6: $F_p \leftarrow \text{Conv}(K, F_\mathcal{X})$ ▷ Finally, typical convolution between K and $F_\mathcal{X}$

每一个X-conv针对p的输入是S = {(pi , fi) : pi ∈ N}, 注意这里S是一个无序集合。不失一般性，S可以表示为KxD的矩阵，P = (p1,p2, ...,pK)T 和一个KxC1的矩阵F = (f1, f2, ..., fK)T。（一个是点的位置矩阵，一个是特征的矩阵），所以X-Conv的参数有K × (C1 + Cδ) × C2个。

- 1 P′ ← P − p 得到P相对于p的坐标
- 2 Fδ ← MLPδ (P′) 将每个点映射到Cδ维的空间中，逐点使用MLP
- 3 F∗ ← [Fδ , F] 把Fδ 和 F拼接起来 F∗ 是一个 K × (Cδ + C1) 矩阵
- 4 X ← MLP(P′) 用P′预测KxK矩阵
- 5 FX ← X × F∗ 到这一步就做完X变换了
- 6 Fp ←Conv(K, FX) 做卷积

Algorithm

$$
\begin{aligned}
F_p &= \mathcal{X}\text{-}Conv(K, p, P, F) \\
&= Conv(K, MLP(P - p) \times [MLP_\delta(P - p), F]),
\end{aligned}
$$

Real-Time Continuous Pose Recovery of Human Hands Using Convolutional Networks

1） Detect and Segment Hand

2） Estimate Pose

3） Validate or Refine

challenge

1）手分辨率低（Low-res hand）

2）背景杂乱（Clutter background）

3）手与其它对象交互（Object/surface interaction）

4）手被遮挡（Occlusions/Self-occlusions）

5）不同手势相似（Self-similarity）

6）多自由度（many DoF(Degree of Freedom)）

7）多视角（Multiple viewpoints）

8）不同的形状和尺寸

本文解决的问题

- 方案延迟小（一帧）
- 对手被遮挡（self-occlusions）有很强的鲁棒性
- 对于不同手势相似（self-similarity）有较好区分度
- 无需额外的markers
- 可以对人手大小在307kpx的尺寸进行处理

流程

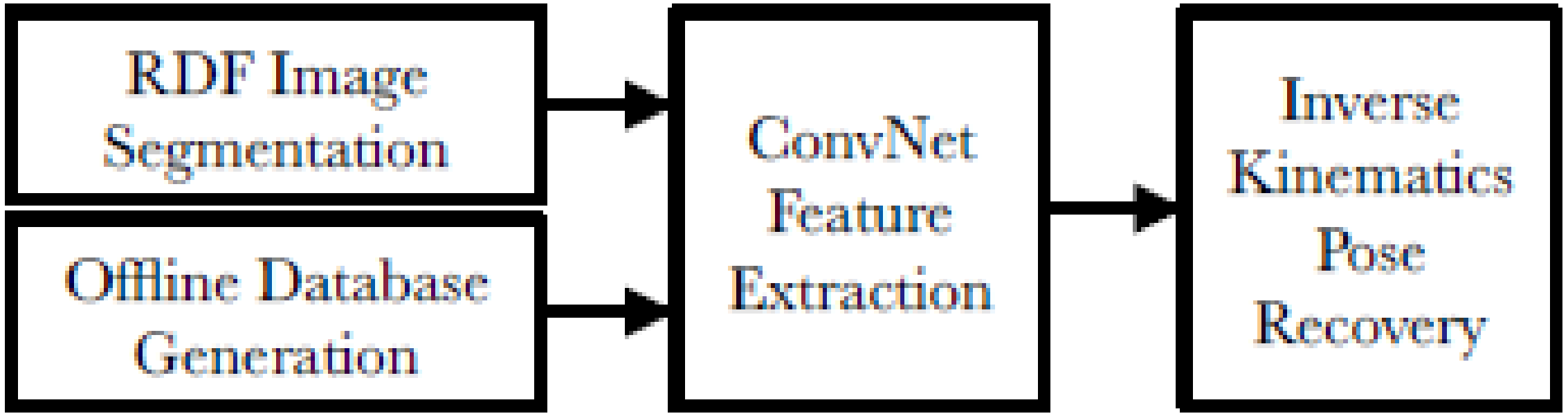


Fig. 1: Pose Recovery Pipeline Overview

Part1： RDF(Random Decision Forest)

(a) Ground-Truth Labels (b) Labels Inferred by RDF

Fig. 2: Decision Forest Data: learned labels closely match target

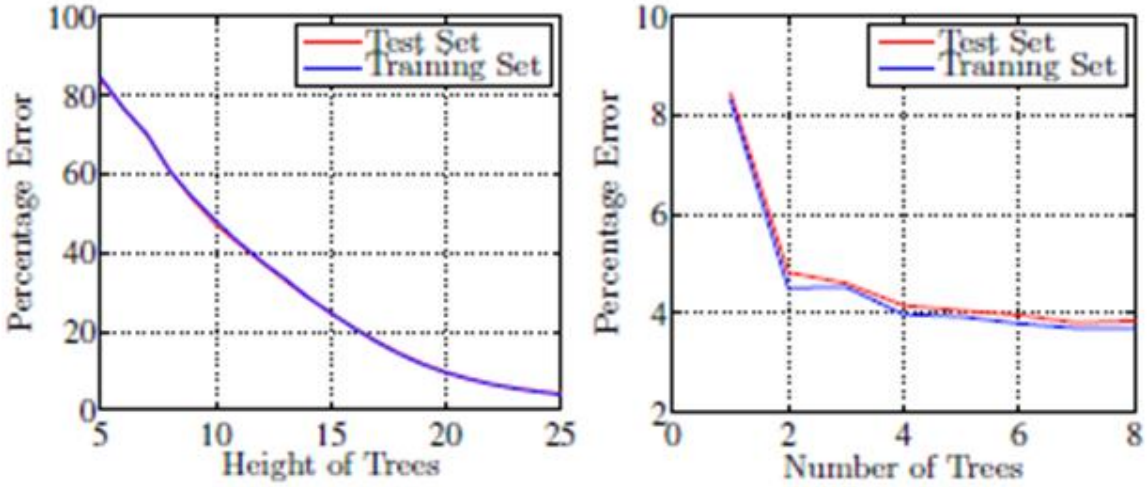


Fig. 10: RDF Error

选择高度25，数量4的随机森林能使准确度和速度达到最佳平衡。

part2: heatmap

使用卷积网络提取14个关节点的heatmap

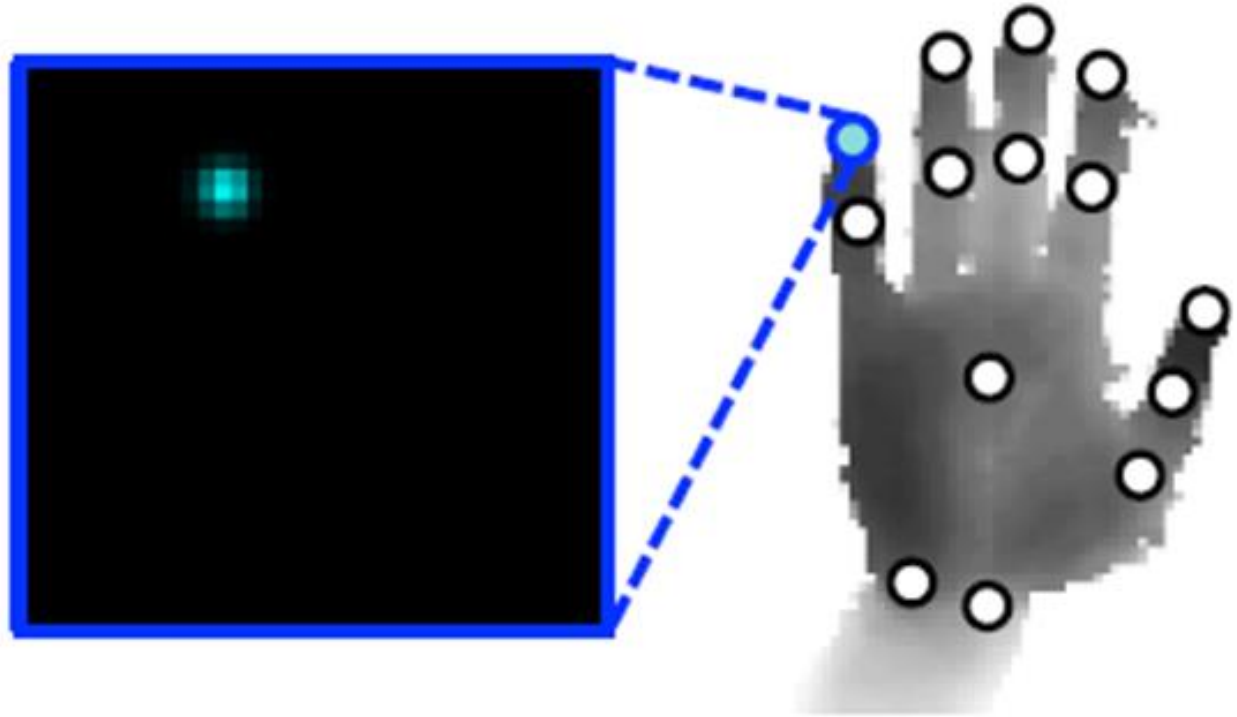
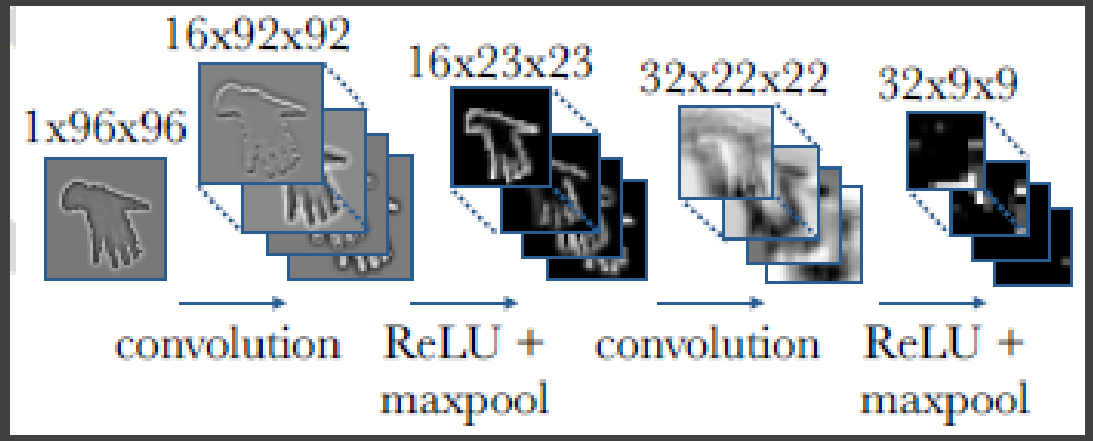


Fig. 5: Depth image overlaid with 14 feature locations and the heat-map for one fingertip feature.

Cont’d

- 通过RDF输出的手部定位信息以三种不同的分辨率（96x96px, 48x48px, 24x24px）输入到二层神经网络，获得相应的heatmap
- 以下是96x96px分辨率输入卷积网络获得特征图的过程：



在获取三种分辨率下的特征图后，经过两层神经网络获取heat-map：

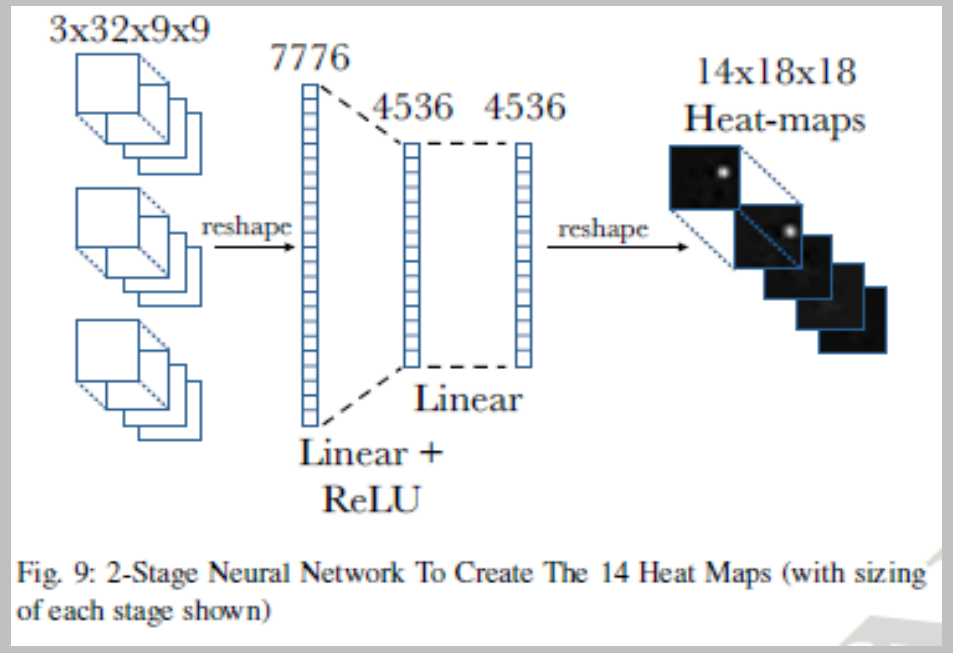


Fig. 9: 2-Stage Neural Network To Create The 14 Heat Maps (with sizing of each stage shown)

权值更新超参数

$$\Delta w_i = \gamma \Delta w_{i-1} - \lambda \left(\eta w_i - \frac{\partial L}{\partial w_i} \right)$$

$$w_{i+1} = w_i + \Delta w_i \tag{3}$$

Where w_i is a bias or weight parameter for each of the network modules for epoch i (with each epoch representing one pass over the entire training-set) and $\frac{\partial L}{\partial w_i}$ is the partial derivative of the error function L with respect to the learnable parameter w_i averaged over the current batch. We use a constant learning rate of $\lambda = 0.2$, and a momentum term $\gamma = 0.9$ to improve learning rate when close to the local minimum. Lastly, an L2 regularization factor of $\eta = 0.0005$ is used to help improve generalization.

Part3: dataset

- 使用Primesense Carmine 1.09(结构光)抓取 RGB-D数据(每一帧的关节位置通过3个Kinect获取)
- 使用PSO(粒子群算法)回归出dataset的ground truth
- 因为PSO在接近极值的时候收敛变慢，本文在它之后加上一个Nelder-Mead优化算法
- 72K训练样本(1人)，8K测试帧(2人)

Part4: Inverse Kinematics

$$f(m) = \sum_{i=1}^{n} [\Delta_i(m)] + \Phi(C) \tag{4}$$

$$\Delta_i(m) = \begin{cases} \left\| (u,v,d)_i^t - (u,v,d)_i^m \right\|_2 & \text{If } d_i^t \neq 0 \\ \left\| (u,v)_i^t - (u,v)_i^m \right\|_2 & \text{otherwise} \end{cases}$$

$$\Phi(C) = \sum_{k=1}^{n} w_k [\max(C_k - C_{k\max}, 0) + \max(C_{k\min} - C_k, 0)]$$

- In case this UV location lies on a depth shadow where no depth is given in the original image, we store the computed 2D image for this point in the original image space. Otherwise, we store its 3D point.
- ϕ(C) enforces a soft constraint that coefficient values stay within a predetermined range (Cmin and Cmax)

result

(a) (b) (c)

Fig. 14: Real-Time Tracking Results, a) Typical Hardware Setup, b) Depth with Heat-Map Features, c) ConvNet Input and Pose Output

泛化性

Fig. 16: Hand Shape/Size Tolerance: RGB ground-truth (top row), depth with annotated ConvNet output positions (bottom row)

Hand PointNet

主要贡献：

第一篇使用pointNet解决 Hand Pose Estimation的文章，对比传统的3D CNN方法效率更高。

使用 oriented bounding box的方法在obb里将输入点云坐标归一化，应对输入与输出旋转有关产生识别不准确的问题。

增加指尖修正网络，使识别准确率更高。

运行时间：20.5ms 模块大小：10.3MB（对比3D CNN网络420MB）

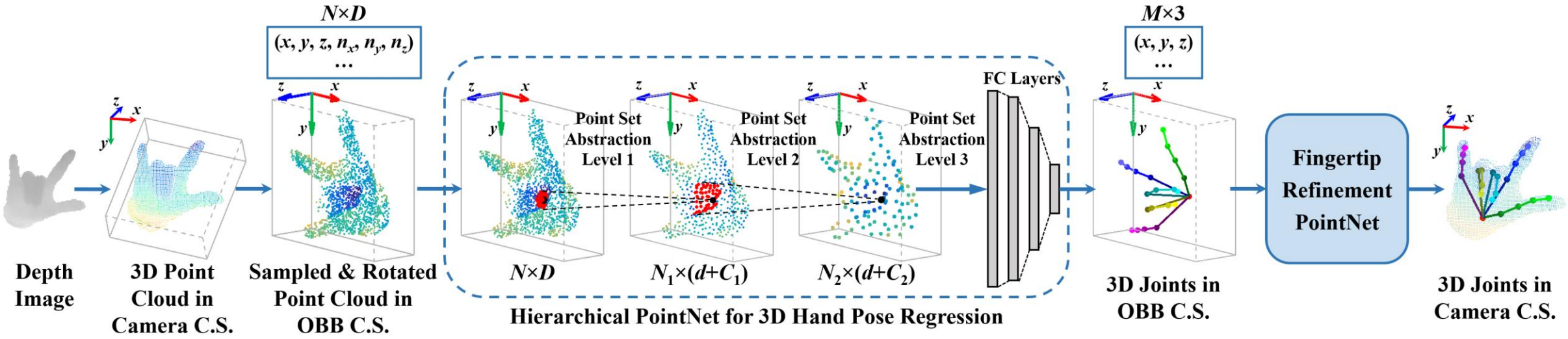
Hand PointNet VS traditional 3D CNN

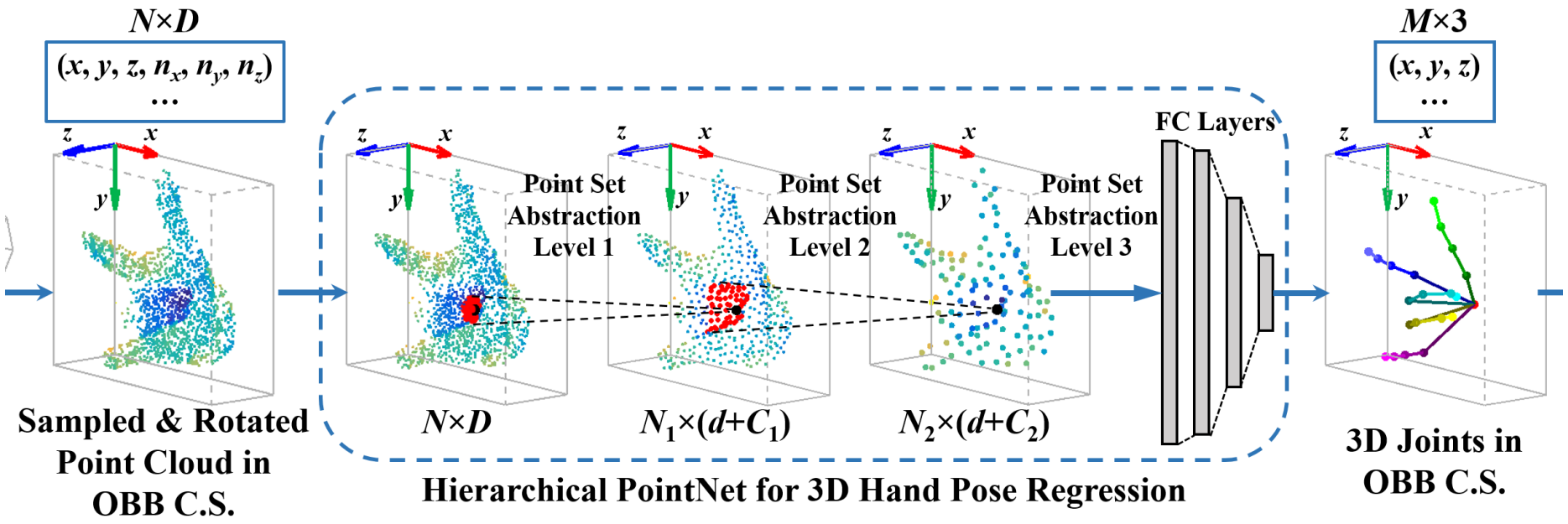
传统3D CNN网络使用深度图像3D像素化后的方式进行与2D CNN类似的卷积处理。但是众所周知:

①多加一个维度需要增加一个维度的参数，这样的叠加会使得3D CNN 网络处理速度变的很慢，相对的，设备要求很高。

②三维空间体素化（Volumetric）后会存在很多空间上的浪费。

Hand PointNet

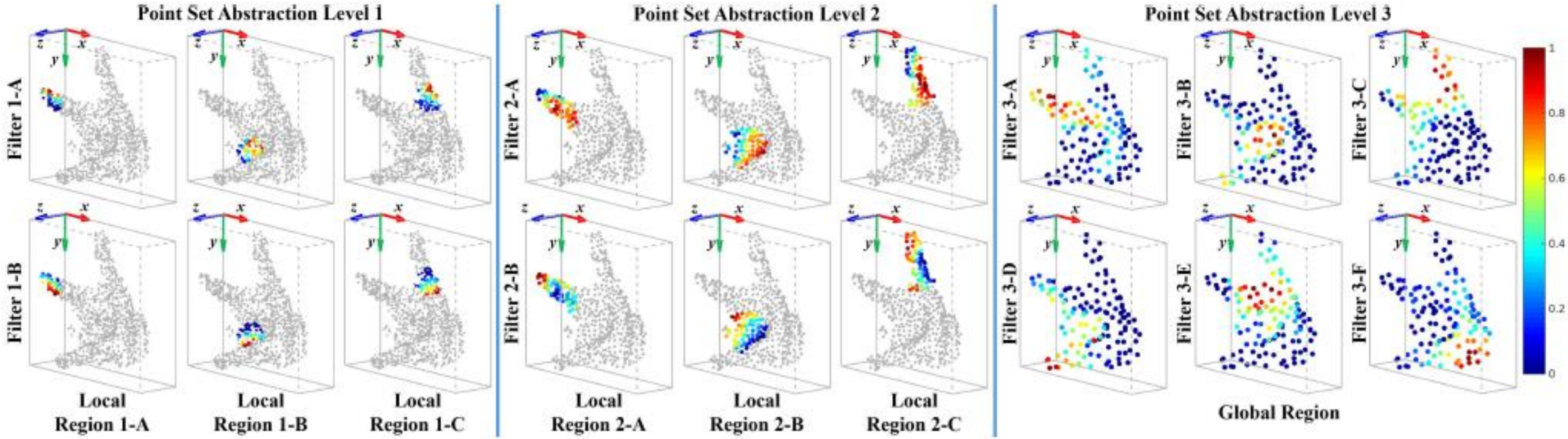




Hand PointNet part1: Modified PointNet

- 传统pointNet网络的局限性在于，无法层级式的获得局部图像特征。
- 采用 Hierachical PointNet，在每层不同的局部使用共享权值MLP。在最后层级提取global point cloud feature。

Cont’d

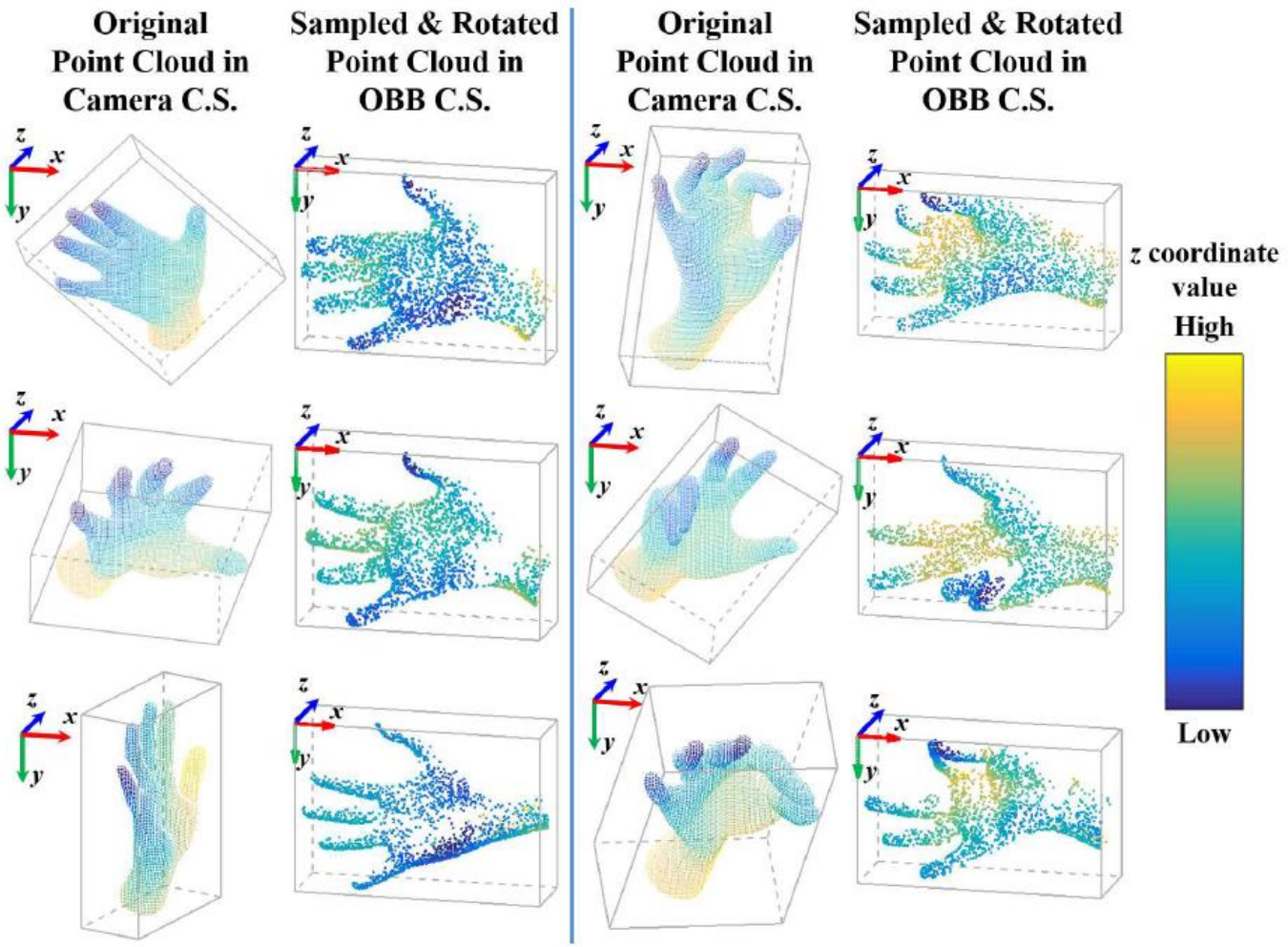

Point Set Abstraction Level 1
Point Set Abstraction Level 2
Point Set Abstraction Level 3
Filter 1-A
Filter 1-B
Filter 2-A
Filter 2-B
Filter 3-A
Filter 3-B
Filter 3-C
Filter 3-D
Filter 3-E
Filter 3-F
Local Region 1-A
Local Region 1-B
Local Region 1-C
Local Region 2-A
Local Region 2-B
Local Region 2-C
Global Region
1
0.8
0.6
0.4
0.2
0

Hand PointNet part2: Oriented Bounding Box

在手势识别中存在一个待解决的问题：由于输出的手势与输入的手势旋转方向有关，我们不能使用basic pointNet中STN的方法找到X-transform matrix并通过maxpooling消除旋转特征。

为解决这个问题，本文在输入Hierarchical pointNet之前添加了一个预处理归一化模块，在该模块中将输入的点云图通过PCA进行主成分分析，并进行归一化到一个 oriented bounding box中。在得到最后的手关节结果后还原模型位置，再进行匹配。

Cont’d


Original Point Cloud in Camera C.S.
Sampled & Rotated Point Cloud in OBB C.S.
Original Point Cloud in Camera C.S.
Sampled & Rotated Point Cloud in OBB C.S.
z coordinate value
High
Low

formula

$$\begin{aligned} \boldsymbol{p}^{obb} &= \left(\boldsymbol{R}_{obb}^{cam}\right)^{T} \cdot \boldsymbol{p}^{cam}, \\ \boldsymbol{p}^{nor} &= \left(\boldsymbol{p}^{obb} - \bar{\boldsymbol{p}}^{obb}\right) / L_{obb}, \end{aligned} \tag{1}$$

where $\boldsymbol{R}_{obb}^{cam}$ is the rotation matrix of the OBB in camera C.S.; $\boldsymbol{p}^{cam}$ and $\boldsymbol{p}^{obb}$ are 3D coordinates of point $\boldsymbol{p}$ in camera C.S. and OBB C.S., respectively; $\bar{\boldsymbol{p}}^{obb}$ is the centroid of point cloud $\left\{\boldsymbol{p}_i^{obb}\right\}_{i=1}^{N}$; L_{obb} is the maximum edge length of OBB; $\boldsymbol{p}^{nor}$ is the normalized 3D coordinate of point $\boldsymbol{p}$ in the normalized OBB C.S.

$$\hat{\boldsymbol{\phi}}_m^{cam} = \boldsymbol{R}_{obb}^{cam} \cdot \left(L_{obb} \cdot \hat{\boldsymbol{\phi}}_m^{nor} + \bar{\boldsymbol{p}}^{obb}\right). \tag{2}$$

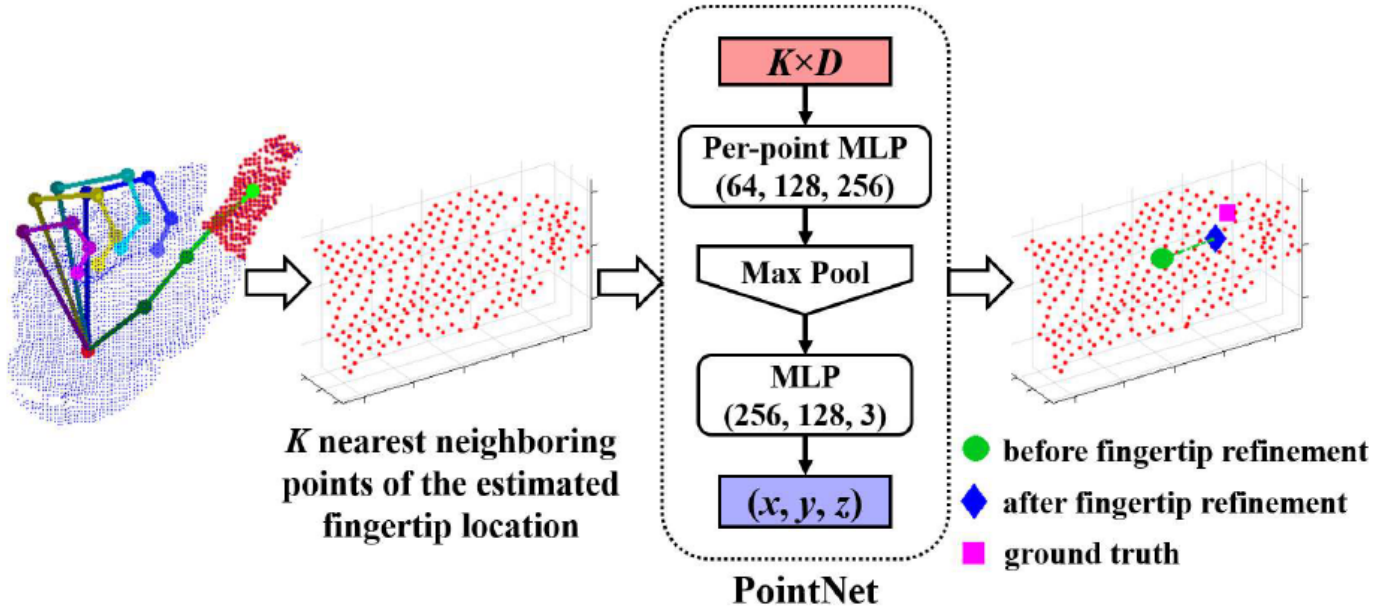
Hand PointNet part3: Hand Pose Regression Network

- Input: $\boldsymbol{X}^{nor} = \{\boldsymbol{x}_i^{nor}\}_{i=1}^{N} = \{(\boldsymbol{p}_i^{nor}, \boldsymbol{n}_i^{nor})\}_{i=1}^{N}$
- 三层结构：first 2 levels: N1 = 512, N2 = 128 local regions, each local region contains k=64 points, extracts C1 = 128, C2 = 256 dimensional features for each local regions respectively. Last level: 1024-dim global feature vector which is mapped to an F-dim(F < 3xM) output vector by three fully-connected layers.提供 $\{(\boldsymbol{X}_t^{nor}, \boldsymbol{\Phi}_t^{nor})\}_{t=1}^{T}$

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{t=1}^{T} \|\boldsymbol{\alpha}_t - \mathcal{F}(\boldsymbol{X}_t^{nor}, \boldsymbol{w})\|^2 + \lambda \|\boldsymbol{w}\|^2 \quad (3)$$

$$\boldsymbol{\alpha}_t = \boldsymbol{E}^T \cdot (\boldsymbol{\Phi}_t^{nor} - \boldsymbol{u}) \qquad \hat{\boldsymbol{\Phi}}^{nor} = \boldsymbol{E} \cdot \mathcal{F}(\boldsymbol{X}^{nor}, \boldsymbol{w}^*) + \boldsymbol{u}. \quad (4)$$

Hand PointNet part4: Fingertip Refinement Network



Cont’d

设计原因：

①指尖的estimation error 一般比其他关节大。

②指尖位置矫正一般比较容易，特别是在手指伸直的情况下。

result

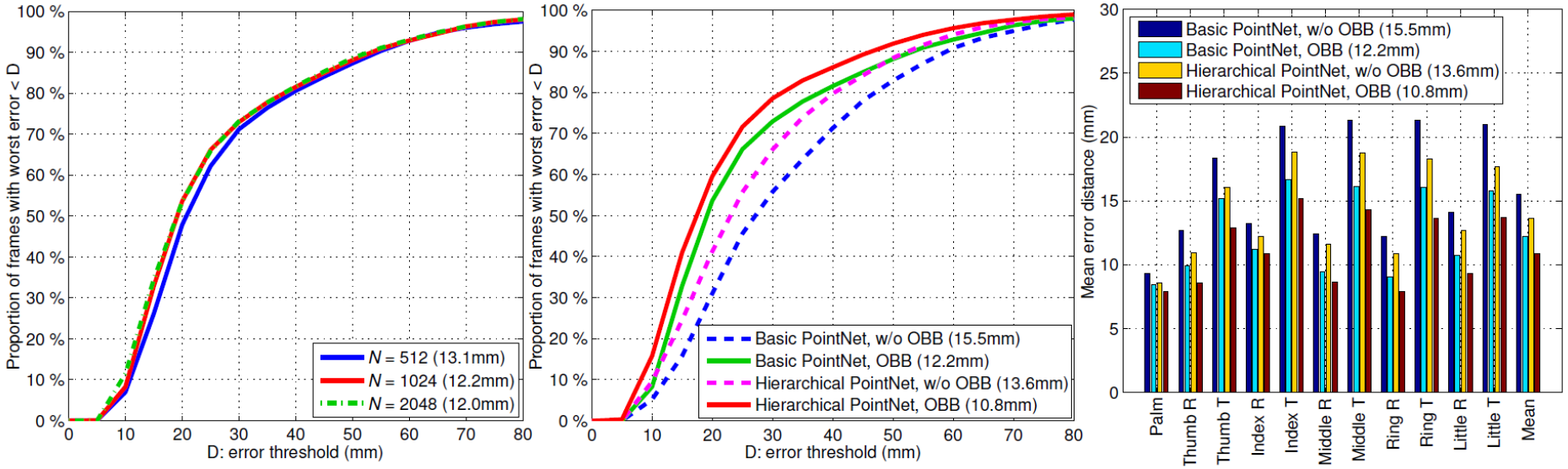


Figure 6: Self-comparison of different methods on NYU dataset [38]. **Left**: the impact of different numbers of sampled points on the proportion of good frames. **Middle & Right**: the impact of different PointNet architectures and normalization methods on the proportion of good frames as well as on the per-joint mean error distance (R: root, T: tip). The overall mean error distances are shown in parentheses.

Cont’d

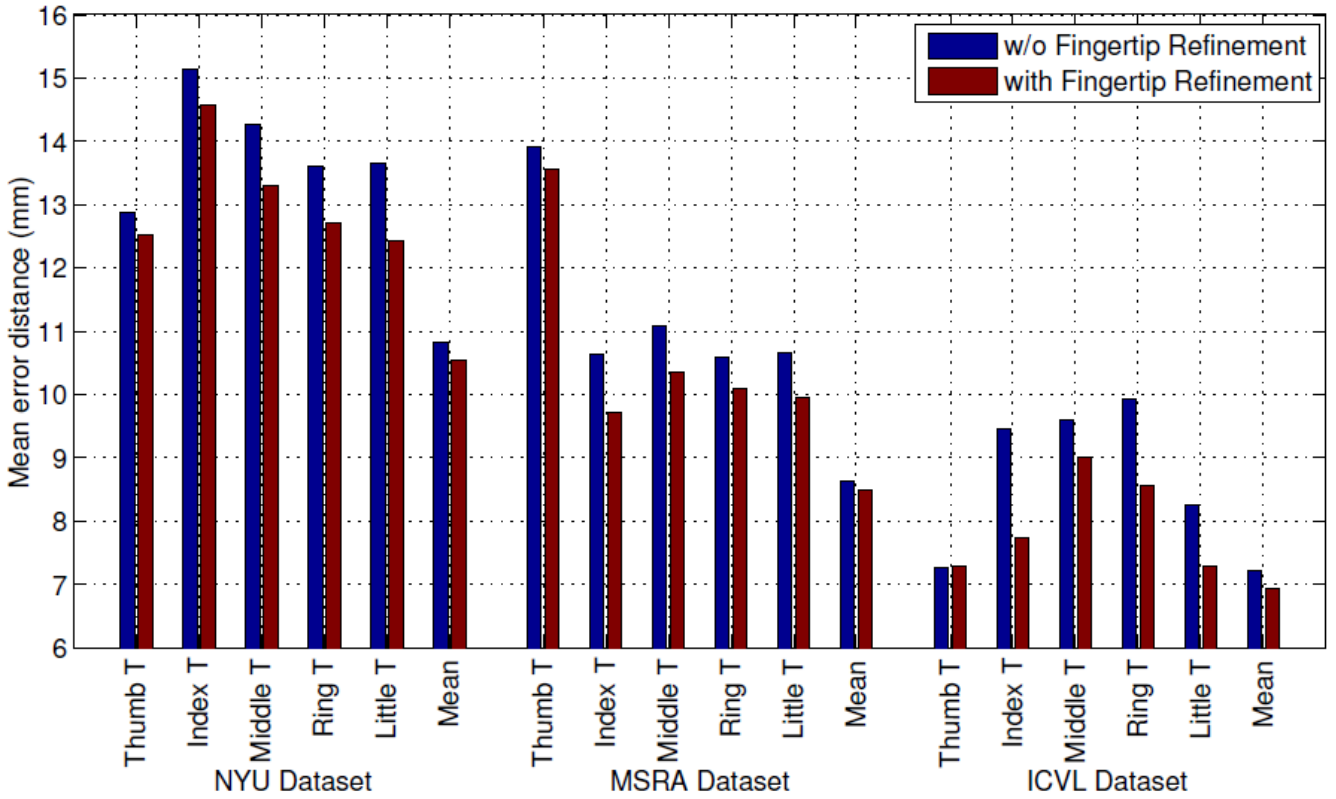


Figure 7: The impact of fingertip refinement on fingertips’ mean error distances and the overall mean error distance on NYU [38], MSRA [33] and ICVL [34] datasets (T: tip).

Cont'd

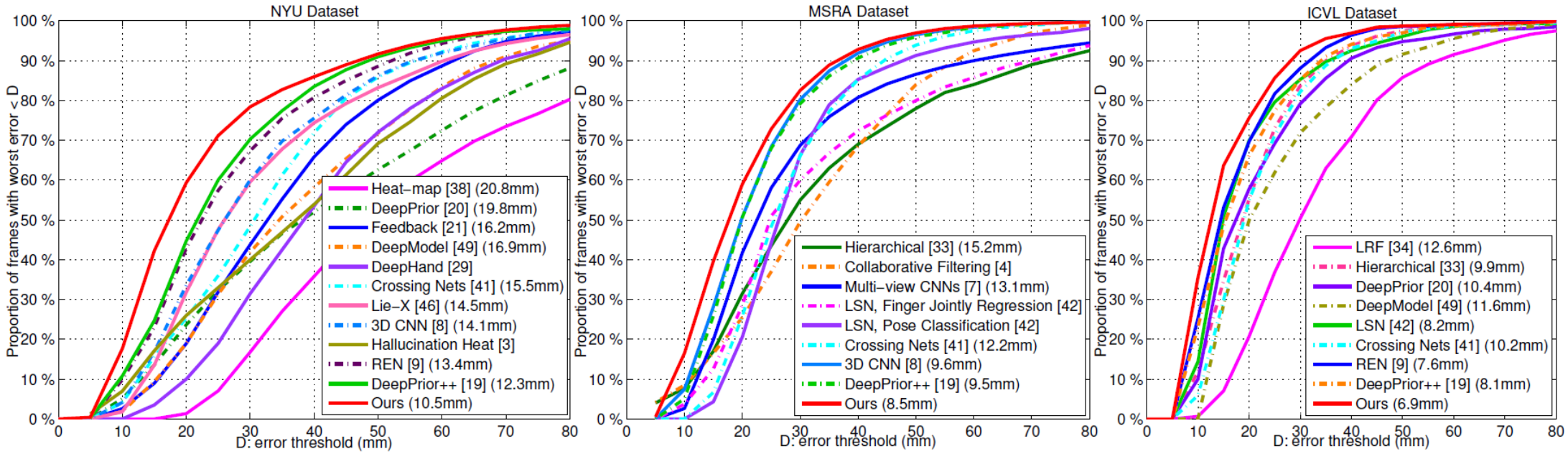


Figure 8: Comparison with state-of-the-art methods on NYU [38] (left), MSRA [33] (middle) and ICVL [34] (right) datasets. The proportions of good frames and the overall mean error distances (in parentheses) are presented in this figure.

Cont'd

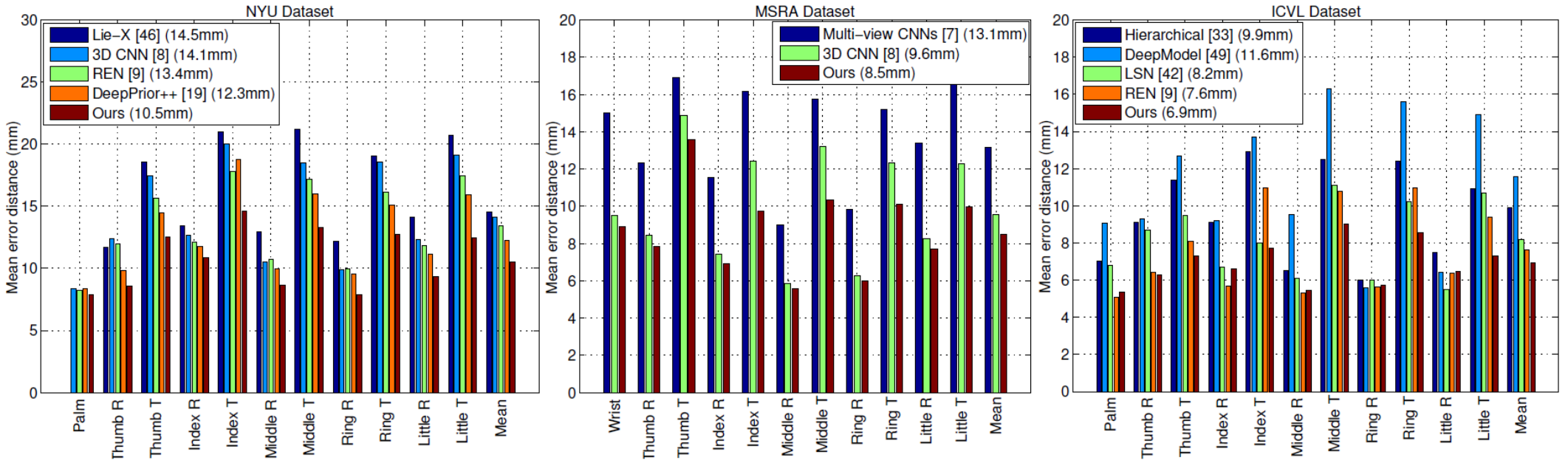


Figure 9: Comparison with state-of-the-art methods on NYU [38] (left), MSRA [33] (middle) and ICVL [34] (right) datasets. The per-joint mean error distances and the overall mean error distances are presented in this figure (R: root, T: tip).

Cont’d

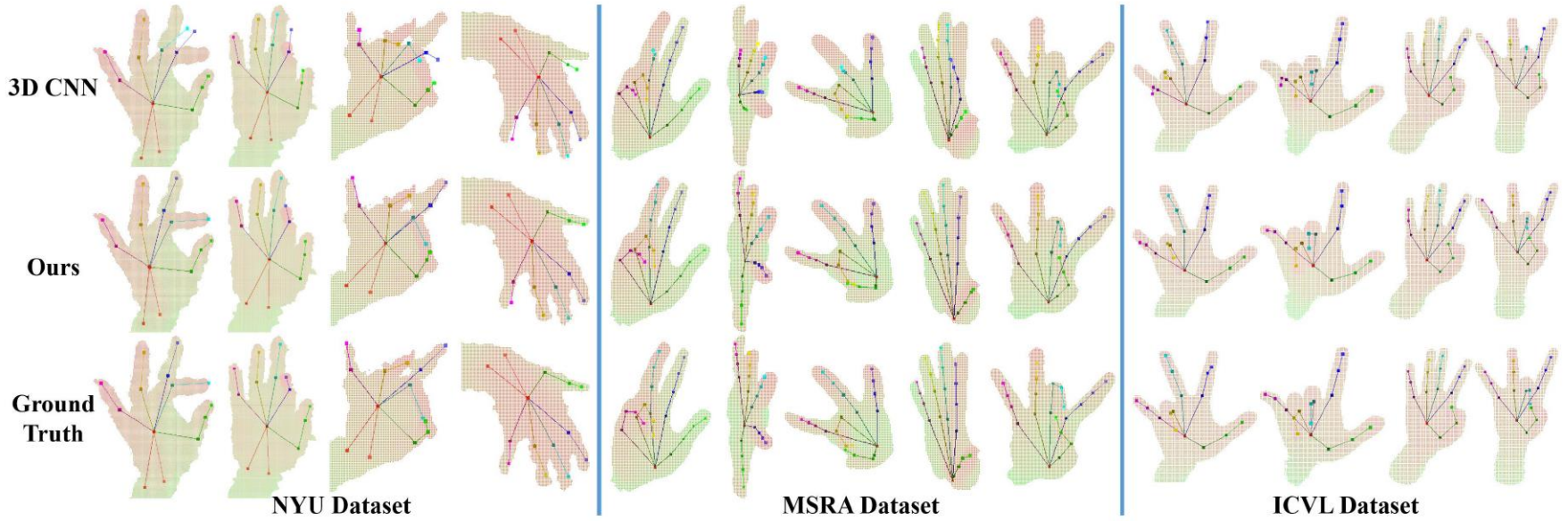


Figure 10: Qualitative results for NYU [38] (left), MSRA [33] (middle) and ICVL [34] (right) datasets. We compare our method (in the second row) with the 3D CNN-based method in [8] (in the first row). The ground truth hand joint locations are presented in the last row. We show hand joint locations on depth images. Different hand joints and bones are visualized with different colors. This figure is best viewed in color.

reference

- **PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation**
- **PointNet++： Deep Hierarchical Feature Learning on Point Sets in a Metric Space**
- **PointCNN: Convolution On X-Transformed Points**
- **Real-Time Continuous Pose Recovery of Human Hands Using Convolutional Networks**
- **Hand PointNet: 3D Hand Pose Estimation using Point Sets**