Abstract

Recently, 3D input data based hand pose estimation methods have shown state-of-the-art performance, because 3D data capture more spatial information than the depth image. Whereas 3D voxel-based methods need a large amount of memory, PointNet based methods need tedious preprocessing steps such as K-nearest neighbour search for each point. In this paper, we present a novel deep learning hand pose estimation method for an unordered point cloud. Our method takes 1024 3D points as input and does not require additional information. We use Permutation Equivariant Layer (PEL) as the basic element, where a residual network version of PEL is proposed for the hand pose estimation task. Furthermore, we propose a voting-based scheme to merge information from individual points to the final pose output. In addition to the pose estimation task, the voting-based scheme can also provide point cloud segmentation result without ground-truth for segmentation. We evaluate our method on both NYU dataset and the Hands2017Challenge dataset, where our method outperforms recent state-of-the-art methods.

1. Introduction

Hand pose estimation plays an important role in human-robot interaction tasks, such as gesture recognition and learning grasping capability by human demonstration. Since the emergence of consumer level depth sensing devices, a lot of depth image based hand pose estimation methods appeared. Many state-of-the-art methods use depth image as input, which provides the conveniences to use the well developed convolutional neural networks or residual networks. However, methods using 2D images as input cannot fully utilize 3D spatial information in the depth image. Furthermore, the appearance of the depth image is dependent on the camera parameters, such that the trained model using one camera’s image cannot generalize well to another camera’s image. On the other hand, 3D data is more “direct” and “distinctive” than depth image because the appearance of 3D data is unique and invariant to camera parameters.

Recently, methods using 3D data as input have shown the outperformance over depth image based methods [36]. One way to use 3D input data is to convert 2D depth image to volumetric representation, such as 3D voxels [15] [4], where occupied 3D voxel is set to 1 and voxels with empty space is set to 0. Using the voxelized data brings the convenience to directly use 3-dimensional CNN learning structure. However, the voxelization requires large amount of memory to represent the input and output data, which prevents the deployment of a very deep structure.

Another way to use 3D input data is to use unordered point cloud as input [6][9][3]. Recently, PointNet, a deep learning structure for point cloud, has shown its success in different tasks. The PointNet estimates point-wise features for individual points and extract global feature from individual points using a max-pooling layer, such that the network is invariant to the order of points. Ge et al. use Point-
We evaluate our method on Hands2017 Challenge dataset and NYU dataset, where state-of-the-art performance is shown. The proposed method achieves the lowest pose error on the Hands2017 Challenge dataset at the time of submission.

2. Related work

A lot of research about hand pose estimation has been done in the last decade, which can be categorized to generative, discriminative and hybrid methods. Generative methods rely on a hand model and an optimization method to fit the hand model to the observations [25][29][23][19]. Discriminative methods use learning data to learn a mapping between observation and the hand pose [17][30][15][4][3][16][26][28]. Hybrid methods use a combination of the generative and discriminative methods [18][27][34]. Our method is a learning based method thus falls into the second category.

Deep learning for hand pose estimation

With the success of deep learning methods for 2D computer vision, depth image based deep learning methods also showed good performance in hand pose estimation task. Tompson et al. use 2D CNN to predict heatmaps of each joint and then rely on PSO optimization to estimate the hand pose [30]. Oberweger et al. [17] uses 2D CNN to directly regress the hand pose out of the image features, where a bottleneck layer was used to force the predicted pose obey certain prior distribution. In a later work, Oberweger [16] replaced CNN to a more sophisticated learning structure, ResidualNet50, to improve the performance of feature extraction. Zhou et al. [40] regress a set of hand joint angles and feed the joint angles into an embedded kinematic layer to obtain the final pose. Ye et al. [33] use a hierarchical mixture density network to handle the multi-modal distribution of occluded hand joints.

Recently, 3D deep learning has been also applied for the hand pose estimation task. Moon et al. use 883 voxels to represent hand’s 3D geometry and use 3D CNN to estimate hand pose [15]. Their method achieved very accurate result, however, 3D voxelization of the input and output data requires large memory size, such that their method only runs at 3.5 FPS. Ge et al. [6][9] use 1024 3D points as input, and rely on PointNet [22] structure to regress the hand pose. Their method achieved satisfying performance, but tedious pre-processing steps are required, which includes oriented bounding box (OBB) calculation, surface normal estimation and k-nearest-neighbours search for all points. Chen et al. improves Ge’s method by using a spatial transformer network to replace the OBB and furthermore added a auxiliary hand segmentation task to improve the performance [3]. Their method can be trained end-to-end without OBB, but the segmentation ground-truth data require a extra pre-computation step from the pose data.

3D Deep learning

Since 3D data cannot be directly fed into a conventional 2D CNN, some methods project the 3D data onto different views to obtain multiple depth images and perform CNN on all images [7][21][10][35]. Another way to process the 3D data is to use volumetric representation and process the data with 3D CNN [8][32][14][15]. These methods can capture the feature of input data more effective, but they require large memory size. Qi et al. developed PointNet to handle unordered point cloud [20]. The PointNet estimates point-wise local features and obtains global features with a max-pooling layer. Later on, PointNet++ extends PointNet by hierarchically upsampling the local features into higher levels [22].
Other recent methods taking 3D points as input include point-wise CNN [11], Deep kd-Networks [12], Self-Organizing Net [13] and Dynamic Graph CNN [31]. Despite their good performance for different tasks, they all require extra steps to estimate k-nearest neighbours or construct kd-tree, which are not required in our proposed residual PEL network.

3. Methodology

The overview of our method is illustrated in Fig. 2. Our method takes $N$ 3D points $P \in \mathbb{R}^{N \times 3}$ with arbitrary order as input, and outputs the vectorized 3D hand pose $y \in \mathbb{R}^J$ in the end, where $J = 3 \times \#\text{joints}$. To estimate the hand pose, the residual permutation equivariant layers (PEL) (Fig. 4) first extract features from each point (Section 3.2). Using the point-wise local features, we use point-to-pose voting to estimate the final pose output (Section 3.3), where two versions for point-to-pose voting are developed, which are the detection version and the regression version.

3.1. Pre-processing with view normalization

For pre-processing, first, the depth pixels in the hand region are converted to 3D points. The next step is to create a 3D bounding box for the hand points to obtain normalized coordinate of these points. A usual pre-processing method will simply create a bounding box aligned with the camera coordinate system (Fig 3a). However, because of self-occlusion of the hand, this will result in different set of observation points for the exact same pose label, which creates one-to-many mapping of the input-output pairs.

To maintain the one-to-one mapping relation of the input-output pairs, we propose to use view normalization to align the bounding box’s z-axis $[0, 0, 1]^T$ with the view direction towards the hand centroid point $c \in \mathbb{R}^3$. The alignment is performed by rotating the hand points with a rotation matrix $R_{\text{cam}}$:

$$
\begin{align}
\alpha_y &= \tan^{-1}(c_y, c_z), \\
\bar{c} &= R_y(-\alpha_y) \cdot c, \\
\alpha_x &= \tan^{-1}(\bar{c}_y, \bar{c}_z), \\
R_{\text{cam}} &= R_y(-\alpha_y) \cdot R_x(\alpha_x).
\end{align}
$$

After rotating the observation points and ground truth pose with $R_{\text{cam}}$, the hand is rotated such that it appears right in front of the camera. As illustrated in Fig. 3b, the one-to-many mapping problem is then avoided.

3.2. Residual Permutation Equivariant Layers

The feature extraction module in our method is called Residual Permutation Equivariant Layers. The basic element is the permutation equivariant layer (PEL), which follows the design from [24]. A PEL takes a set of unordered
In this way, each point in the input point cloud is used in our method.

Figure 4. Residual network of permutation equivariant layer

Points: \( P \in \mathbb{R}^{N \times 3} \)

Residual block (3x64)

Residual (64x256)

Residual block (256x1024)

Features: \( F \in \mathbb{R}^{N \times 1024} \)

The detailed proof of the invariance for PEL can be found in [24].
where $b - 0.5$ represents the bin center position.

**Regression version**

In the regression version (Fig. 2 right), each point will directly predict the pose without the intermediate distribution detection. Similarly to the detection version, two separate fully connected modules are used to estimate the importance term $G \in \mathbb{R}^{N \times J}$ and the point-to-pose estimates $\hat{y} \in \mathbb{R}^{N \times J}$. Then the final pose output is merged as the weighted average over all points’ predictions:

$$y_j = \frac{\sum_{n=1}^{N} (G_{nj} \hat{y}_{nj})}{\sum_{n=1}^{N} G_{nj}}. \quad (6)$$

### 3.4. Segmentation using importance term

The importance term $G \in \mathbb{R}^{N \times J}$ is estimated automatically without the ground-truth information. However, it still provides vital information of each point’s importance to the pose output. Therefore, the obtained importance term can be also used for the hand segmentation task based on the most contributed pose dimension. For the $n$-th point having the importance terms $g = G_n$, the point’s most contributed pose dimension is:

$$j_{\text{max}} = \arg \max_j g_j,$$

where the pose dimension $j_{\text{max}}$ can be categorized to a specific hand part. In this work, we categorized the $J$ pose dimensions to palm, thumb, index, ring and pinky fingers.

### 3.5. Training Loss

The only training loss for the detection version is the logarithm loss of the pose distributions:

$$L_{\text{det}} = -\sum_{j=1}^{J} \sum_{b=1}^{B} D_{bj}^{\text{gt}} \log(D_{bj} + \epsilon)$$

$$+ (1 - D_{bj}^{\text{gt}})(1 - \log(D_{bj} + \epsilon)),$$

where $\epsilon = 10^{-7}$ is a small offset to avoid feeding zero to the logarithm operator.

The only training loss used for the regression version is the L2 loss between predicted pose and ground-truth pose:

$$L_{\text{reg}} = \frac{1}{2} \sum_{j=1}^{J} (y_j^{\text{gt}} - y_j)^2. \quad (8)$$

For both detection and regression versions, the importance term $G \in \mathbb{R}^{N \times J}$ is estimated automatically without the ground-truth information.

### 4. Experiment and result

Our hand pose estimation method is evaluated on the Hands2017Challenge dataset [37] and the NYU [30] dataset. The Hands2017Challenge is composed from parts of the Big Hand 2.2M dataset [38] and the First-person Hand Action Dataset (FHAD) [5]. It is currently the largest dataset available. Its training set contains 957032 depth images of five different hands. The test set consists of 295510 depth images of ten different hand shapes, of which five are the same as in the training set and five are entirely new. The NYU dataset contains 72757 training images of a single subject’s hand and 8252 test images that include a second hand shape besides the one from the training set. The NYU dataset provides depth images from three different views, we trained our method both using only frontal view data and using all three views. And we test only using the frontal view.

Our method is implemented using TensorFlow [1]. The networks are trained on a PC with an AMD FX-4300/Intel Core i7-860 CPU and an nVidia GeForce GTX1060 6GB GPU. We train 100 epochs for the NYU dataset and train only 20 epochs for the Hands 2017 challenge dataset since the challenge dataset has a large size. For both datasets, the first 50% of the epochs are trained with smaller number of points ($N = 256$) to boost the training speed. The remaining epochs are trained with a point size of $N = 512$. We used Adam optimizer for training with an initial learning rate of $10^{-3}$ and we decrease the learning rate to $10^{-4}$ for the last 10% of the epochs. For the detection version, we set $r = 15mm$ and $B = 60$. Online augmentation was performed with random translation in all three dimensions within $[-15, 15]mm$, random scaling within $[0.85, 1.15]$ and random rotation around z-axis within $[-\pi, \pi]$.

### 4.1. Evaluation metrics

For the NYU dataset, two standard metrics are used to evaluated the performance. The first metric is the mean joint error, which measures the average Euclidean distance error for all joints across the whole test set. The second metric is correct frame proportion, which indicates the proportion of frames that have all joints within a certain distance to ground truth. The second metric is considered as more difficult since single joint violation will cause an unqualified frame. For the Hands2017Challenge dataset, only the mean joint error is used since the ground-truth data of test set is not publicly available and the official test website only provides the mean joint error result.

### 4.2. Self-comparison

In this subsection, we perform self-comparison to show the effects of different components in our method. The detailed comparison can be found in Table 1.


\[\text{Table 1. Self-comparison result}\]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>256 points</td>
<td>11.34</td>
<td>10.23</td>
<td>11.93</td>
<td>11.67</td>
<td>9.93</td>
<td>11.67</td>
<td>9.93</td>
</tr>
<tr>
<td>512 points</td>
<td>13.14</td>
<td>11.17</td>
<td>12.04</td>
<td>11.21</td>
<td>10.11</td>
<td>9.82</td>
<td>10.82</td>
</tr>
<tr>
<td>1024 points</td>
<td>11.21</td>
<td>9.82</td>
<td>11.67</td>
<td>9.82</td>
<td>9.25</td>
<td>8.97</td>
<td>8.97</td>
</tr>
<tr>
<td>2048 points</td>
<td>11.21</td>
<td>9.87</td>
<td>11.67</td>
<td>9.87</td>
<td>9.32</td>
<td>9.08</td>
<td>8.35</td>
</tr>
</tbody>
</table>

**Figure 5.** Comparison with state-of-the-arts on NYU [30] dataset. Left: mean errors of different joints. Right: proportion of correct frames based on different error thresholds.

**Table 2. Comparison of our method with state-of-the-art methods on the Hands2017Challenge dataset**

|----------|---------------------------|----------------|------------------|-------|--------------|------------------|----------|------------------------|--------------|---------------|

**Table 3. Comparison of our method with state-of-the-art methods on the NYU dataset**

**View normalization.** To validate the necessity of view normalization, we trained our method using both view normalized data and original data for the detection version. It is evident from Table 1 that view normalization decreases the pose estimation error by about 1.5 mm for the Hands2017Challenge dataset.

**Detection vs. regression.** Yuan et.al. indicates that detection based methods work in general better than regression-based methods [36], therefore we implemented both detection-based (ours/distribution) and regression-based (ours/regression) variations. As seen from Table 1, in both datasets, both variations show similar performance, where regression-based variation slightly outperforms the detection-based counterpart. Possible reasons for this can be quantization effect of the binary distribution and the simplification of 1-dimensional heat vectors compared to 2D or 3D heat maps used in previous works. However, the 1D heat vector representation is much more efficient than the 3D heatmap representation. For the heat vectors, we need \(B \times J\) values to represent the pose output, whereas 3D heatmaps require \(J \times B^3\) values [15]. In future work, it is worth to investigate more different loss types and heat map representations.

**Number of points.** Taking advantage of the PEL structure and voting-based scheme, our method is very flexible to the input point cloud size. Although the network was trained with 512 points, arbitrary number of points can be used at the testing stage. For an online application, this property can be beneficial to choose an arbitrary number of points based on the computational resources available. As seen from Table 1, different number of points were tested.
Figure 6. Segmentation results based on importance weights (best viewed in color). Points: input point cloud, color indicates depth value, blue points are more distanced and red points are more closer to the camera. Segmentation: each part of the hand is indicated with an different color, palm (red), thumb (green), index (blue), middle (yellow), ring (cyan), pinky (pink) and irrelevant points with low importance weight for all parts (gray).

for both datasets. Our method can achieve good performance with only 256 points, the mean joint error only increased by 0.11 mm compared to 512 points. In general, more points provides better performance, but it doesn’t improve any more after 1024 points. Therefore, we choose 1024 points for testing to compare our method with other state-of-the-art methods.

4.3. Comparison to state-of-the-art methods

Hands2017Challenge dataset. Since the ground-truth data for the testing set publicly available, some previous papers divide the training set on their own to create their own testing set. Therefore, for fair comparison, we only compare to those methods, who have also tested on the official testing website\(^2\). In Table 2, we compare our method with five other top performing methods on the Hands2017Challenge dataset, which include both methods using 3D input data and methods using 2D depth image. RCN-3D [36], THU VCLab [2] and Vanora [36] use depth image as input data. V2V-PoseNet [15] uses voxel representation for both input data and output heatmaps. Oasis [6] also uses 3D point cloud as input and their method is constructed based on PointNet [20]. Three different errors are used for comparison: 1) the average across the complete test set (\textit{avg test}), 2) the average across the test set of seen subjects’ hand dur-

\(^2\)https://competitions.codalab.org/competitions/17356#learn_the_de
ing training (seen test), and 3) the average across the test set images of unseen subjects’ hand (unseen test). Currently, our method achieves the lowest overall mean joint error on the test dataset of 9.82 mm. For seen subjects’ hand and unseen subjects’ hand, the mean joint errors are 7.15 mm and 12.04 mm respectively, which shows the generalizability of the proposed method even without regularization on the parameters. In comparison to other 3D data based methods, our method is slightly better than V2V-PoseNet, whereas V2V-PoseNet requires 10 good GPUs to run realtime and our method requires only one moderate GPU. Compared to oasis, which also uses 1024 3D points as input, our method is 1.48 mm better, where oasis requires more input information like surface normal and k-nearest neighbours.

**NYU dataset.** For the NYU dataset, we only compared to recent state-of-the-art methods after 2017. For testing the performance, only the frontal view was used. Following previous works [17][30][9], only 14 joints out of 36 joints provided were used for evaluation. For a fair comparison, we only compared to the methods trained solely on the NYU dataset without additional data. The compared methods include depth image based methods (DeepPrior++ [16], DenseReg [4]), 3D voxel based methods (3DCNN [8], V2V-PoseNet [15]) and point cloud based methods (SHPR-Net [3], HandPointNet [6], Point-to-Point [9]). The comparison is shown in Figure 5, where our method performs comparably good with V2V-PoseNet [15] and Point-to-Point [9], and outperforms all other methods. A closer comparison of the mean joint error value can be found in Table 3, where our method trained with single view is the second best, and our method trained with three views outperforms all recent state-of-the-art methods.

### 4.4. Segmentation using importance term

Besides showing the quantitative results relying on the ground-truth data, we also show some qualitative result of the segmentation using the automatically inferred importance term. As seen from Figure 6, the segmentation result is shown alongside the original point cloud. The samples are taken from the Hands2017Challenge dataset. Both samples with all visible fingers and samples with different levels of self-occlusion are shown. In all cases, the fingers are clearly segmented with each other, even the fingers are twisted together. The points has no contribution to any joint has very small importance values and they are classified as background. As Figure 6 shows, the arm and the background points are clearly segmented in gray. Notice that the segmentation result is obtained without the ground-truth data for segmentation. This leads to a future research question about whether we can perform this method on hand-object interaction cases, where the influence of the object can be automatically removed.

### 4.5. Runtime and model size

Compared to depth image based methods, our method requires more computation time and memory storage, this limits our training to use only 512 points (batch size=32) on our hardware setting. To store the learned models, the proposed method takes 38 MB for the regression version and 44 MB for the detection version. Compared to 420MB for a 3D CNN based method [4], our model size is much smaller. For the testing stage, the runtime of our method is 12.5 ms and 10.7 ms per frame for the detection and regression version respectively, where 1024 points are used as input. When less input points is used, the runtime can be further reduced with a small performance loss. Table 4 shows a comparison of runtime to other state-of-the-art 3D methods [15][4][9]. Although the other methods all used a more powerful GPU than ours, our method require the least processing time.

### 5. Conclusion

We propose to use a novel neural network architecture, ResidualPEL, for hand pose estimation using unordered point cloud as input. The proposed method is invariant to input point order and can handle different numbers of points. Compared to previous 3D voxel based methods, our method requires less memory size. And compared to PointNet based methods, our method does not require surface normal and K-nearest-neighbours information. A voting-based scheme was proposed to merge information from individual points to pose output, where the resulting importance term can be also used to segment the hand into different parts. The performance of our method is evaluated on two datasets, where our method outperforms the state-of-the-art methods on both datasets. In future work, the proposed ResidualPEL and voting scheme can be also applied to similar problem such as human pose estimation and object pose estimation.
References


[38] Shanxin Yuan, Qi Ye, Bjorn Stenger, Siddhant Jain, and Tae-Kyun Kim. Bighand2. 2m benchmark: Hand pose dataset and state of the art analysis. In IEEE Conference on Computer Vision and Pattern Recognition, pages 2605–2613, 2017. 5


[40] Xingyi Zhou, Qingfu Wan, Wei Zhang, Xiangyang Xue, and Yichen Wei. Model-based deep hand pose estimation. In