Deep Graphical Feature Learning for the Feature Matching Problem

Zhen Zhang*
Australian Institute for Machine Learning
School of Computer Science
The University of Adelaide
zhen@zzhang.org

Wee Sun Lee
Department of Computer Science
National University of Singapore
leews@comp.nus.edu.sg

Abstract

The feature matching problem is a fundamental problem in various areas of computer vision including image registration, tracking and motion analysis. Rich local representation is a key part of efficient feature matching methods. However, when the local features are limited to the coordinate of key points, it becomes challenging to extract rich local representations. Traditional approaches use pairwise or higher order handcrafted geometric features to get robust matching; this requires solving NP-hard assignment problems. In this paper, we address this problem by proposing a graph neural network model to transform coordinates of feature points into local features. With our local features, the traditional NP-hard assignment problems are replaced with a simple assignment problem which can be solved efficiently. Promising results on both synthetic and real datasets demonstrate the effectiveness of the proposed method.

1. Introduction

Finding consistency correspondences between two sets of features or points, *a.k.a.* feature matching, is a key step in various tasks in computer vision including image registration, motion analysis and multiple object tracking [6, 10, 15, 17, 4]. When effective visual features are available, e.g. from successful deep learning, simple inference algorithms such as the Hungarian/Munkres algorithm [19] can achieve quite good performance [27, 7, 16]. However, for geometric feature matching problems, strong local features are not available; pairwise or higher order features must be applied to find robust matchings. These features, however, will transform the inference problem into the Quadratic/Higher Order Assignment Problem, which is NP-hard in general [2]. Due to the NP-hardness of the inference problem in geometric feature matching, previous works mainly focus on developing efficient relaxation methods [28, 13, 12, 4, 31, 14, 15, 26, 30, 25, 29].

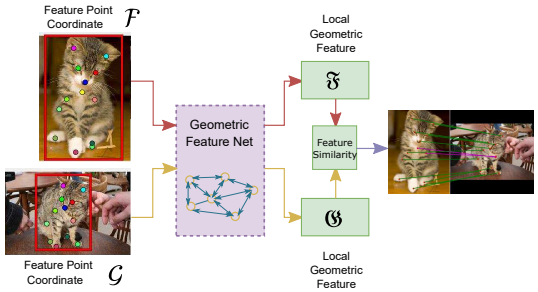


Figure 1: The graph neural netowrk transforms the coordinates into features of the points, so that a simple inference algorithm can successfully do feature matching.

In this paper, we attack the geometric feature matching problem from a different direction. We show that it is possible to learn a mapping from only the point coordinates and the graph structure into strong local features using a graph neural network, such that simple inference algorithms outperform traditional approaches that are based on carefully designed inference algorithms. This process is shown in Figure 1 where point coordinates for the vertices of two graphs are transformed into rich features for effective matching using the Hungarian/Munkres algorithm.

To do the feature matching problem from the point coordinates, various sources of information are likely to be useful. Pairwise information such as length and angle of edges are likely be very important. Multi-scale information may also play a useful role; this can often be obtained with the use of hierarchical structures. Earlier works with hierarchical handcrafted features such as hierarchical structured image pyramid rely on strong local features to construct scale invariant visual features [16, 1, 24]. In deep learning based visual feature matching methods, the hierarchical structured CNN has been applied to get rich local features. It is less clear how to construct features that provide multi-scale information directly from just point coordinates.

*This work is done at the Department of Computer Science, National University of Singapore.

We seek to capture most of these information and transform them into node features that are useful for matching. To do that, we use graph neural networks (GNN). Graph neural network extends normal convolutional neural network (CNN) to irregular graph structured data. Unlike normal CNN, in GNN, each node and its neighbors are associated with different convolution operators. This allows nodes and their neighbours at different relative positions and distances to use different convolution operators, hence providing different edge information. Furthermore, GNNs use message passing algorithms to propagate information, including those at multiple scales, as well as other more global information about the graph structure, to the nodes features. By learning the parameters of the GNN, we can train the network to construct features that are appropriate for the matching task.

Unfortunately, allowing each node and its neighbour to have a different convolution operator requires a large amount of computational resources as well as a large amount of training data for learning. As a result, the general graph neural network suffers from scalability problems. To handle this issue, we proposed an effective method for composing the local convolution kernel for each edge from a global collection of kernels using an attention mechanism. The resulting GNN, named Compositional Message Passing Neural Network (CMPNN), is highly efficient and gives good results in the graph matching problem.

Our main results are summarized as follows:

1. We proposed to use a graph neural network in order to exploit the global structure of a graph for transforming weak local geometric features at points into rich local features for the geometric feature matching problem;
2. With our rich local features, the hard Quadratic/Higher Order Assignment Problem in traditional methods can be downgraded to a simple Linear Assignment Problem, which can be solved by the Hungarian/Munkres algorithm efficiently;
3. We proposed the Compositional Message Passing Neural Network, which uses an attention mechanism to compose a local convolution kernel from a global collection of kernels, enabling us to train and run our network efficiently.

2. Related Works

Geometric Feature Matching Traditionally the geometric feature matching problem is often formulated as a Quadratic/Higher Order Assignment Problem with simple features such as length/angle of edges, angles of triangles and so on [31, 30, 13, 25]. These methods have difficulties with the complexity of inference and the quality of matching. To achieve better accuracy, the order of the features usually needs to be increased. For example, to attain scale and rotation invariance, order 3 features are required and to attain affine invariance, order 4 features are required [28, 12, 14, 26]. Recently, Milan *et al*. [18] used a recurrent neural network to solve these hard assignment problems; however the work was only demonstrated for small scale problem with less than 10 feature nodes.

PointNet The PointNet is a recent geometric feature learning framework for point cloud classification and segmentation [23]. The authors also show that the learned features can be used to establish correspondences between object parts. In the vanilla PointNet, the global feature will be extracted by global pooling and then be propagated to every node. Such a structure is a good match for application such as classification and semantic segmentation. For feature matching, which requires rich local features, such global information propagation procedures may fail due to lack of local information exchange and failure to exploit the hierarchical structure. The point pair feature (PPFnet) extended the PointNet with a hierarchical structure for point cloud feature matching problems [3]. In the PPFnet, given several local point patches, PointNet is used to extract one feature vector for each local patch. Then the features are passed to another PointNet-like network to get the feature representation of each local patch. As the PPFnet strongly depends on PointNet, it still lacks an efficient local information propagation scheme. Thus it requires rich local geometric feature (*i.e*. local point cloud patch) to work. When such features are not available and only the coordinate of the point is given, it is reduced to the PointNet.

Message Passing Neural Network The message passing neural network (MPNN) proposed by Gilmer *et al*. [5] provides a general framework for constructing graph neural networks. We describe a particular instance of MPNN. Assume that we are given a graph organize as adjacency list, a set of node features $\mathbf{x}_i \in \mathbb{R}^{d_{\text{in}}}$ and a set of edge features $\mathbf{e}_{ij}, i \in \{1, 2, \dots, n\}, j \in N(i)$, where $N(i)$ is set of neighbors of node i, include the node itself. In MPNN, the edge features are passed to a neural network to generate a convolution kernel, that is

$$\mathbf{k}_{ij} = h_{\mathbf{k}}(\mathbf{e}_{ij} \,|\, \boldsymbol{\theta}_{\mathbf{k}}), \tag{1}$$

where the kernel $\mathbf{k}_{ij} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ will be used to map the node feature n_j to a d_{out}-dimensional feature vector, and then these features will be gathered to generated new node features, that is [1]

$$\mathbf{y}_i = \underset{j \in N(i)}{\mathfrak{A}} \mathbf{k}_{ij}\, \mathbf{x}_j, \tag{2}$$

where the aggregation operator $\mathfrak{A}$ can be any differentiable set operator that maps a set of d_{out}-dimensional feature vectors to one d_{out}-dimensional feature vector.

[1] To simplify the notation, we omit the activation function throughout the paper, embedding it in $\mathfrak{A}$.

The main drawbacks of vanilla MPNN is the memory consumption. Assume that we are working on a graph with 100 nodes, each node is associated with 10 edges, and $d_{\text{in}} = d_{\text{out}} = 512$, then we will need approximately $100 \times 10 \times 512^2 \times 4 \approx 1$GB memory for storing all $\mathbf{k}_{ij}$ in single float precision. Doing back-propagation with $\mathbf{k}_{ij}$ may require an additional 3-4GB memory. As a result, several MPNN layers can easily exhaust the GPU memory. For large scale data, a more efficient version of MPNN is required.

3. Methodology

3.1. Notations

Given two set of feature points $\mathcal{F} = \{f_i | i = 1, 2, \dots, n\}$ and $\mathcal{G} = \{g_i | i = 1, 2, \dots, n\}$, the feature matching problem aims find an permutation $\pi : \{1, 2, \dots, n\} \mapsto \{1, 2, \dots, n\}$ which maximizes some similarity function

$$\underset{\pi \in \Pi(n)}{\operatorname{argmax}} S([f_{\pi(i)}]_{i=1}^n, [g_i]_{i=1}^n), \tag{3}$$

where $\Pi(n)$ denotes all permutation over set $\{1, 2, \dots, n\}$. Usually the similarity function S can be defined as sum of a series of similarity function with different orders as follows

$$\begin{aligned} &S([f_{\pi(i)}]_{i=1}^n, [g_i]_{i=1}^n) \\ =&\sum_i S_1(f_{\pi(i)}, g_i) + \sum_{ij} S_2([f_{\pi(i)}, f_{\pi(j)}], [g_i, g_j]) + \cdots \end{aligned} \tag{4}$$

In a geometric feature matching problem, the feature f_i and g_i are often coordinates of points. Thus the first order similarity function becomes meaningless and higher order functions must be applied, which requires solving NP-hard Quadratic or Higher Order Assignment Problems.

We address this problem by developing a rich local feature extractor using graph neural network. Now we describe the high level structure of the proposed model. The proposed graph neural network is a set function as follows:

$$\mathfrak{F} = \{\mathsf{f}_i\} = h(\mathcal{F}|\boldsymbol{\theta}), \quad \mathfrak{G} = \{\mathsf{g}_i\} = h(\mathcal{G}|\boldsymbol{\theta}), \tag{5}$$

Then with selected first order similarity function, the correspondence between feature sets can be found via

$$\underset{\pi \in \Pi(n)}{\operatorname{argmax}} \sum_i S_1(\mathsf{f}_{\pi(i)}, \mathsf{g}_i). \tag{6}$$

Loss function Eq. (6) gives the procedure for inference with learned parameter $\boldsymbol{\theta}$. Now we define the loss function for learning using cross-entropy loss. First we define

$$p(\pi(i) = j | \boldsymbol{\theta}) = \frac{\exp(S_1(\mathsf{f}_j, \mathsf{g}_i))}{\sum_j \exp(S_1(\mathsf{f}_j, \mathsf{g}_i))}, \tag{7}$$

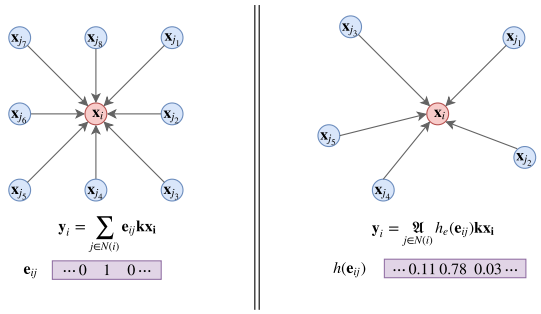


Figure 2: **Left:** Normal CNN. **Right:** Compositional Message Passing Neural Networks (CMPNN). The convolutional kernel in normal CNN and CMPNN can be represented as a $t_e \times d_{\text{out}} \times d_{\text{in}}$ tensor, where t_e is the number of types of edges. While in normal CNN each edge is associated with a particular type represented by the edge feature vector $\mathbf{e}_{ij}$ in one-hot form, in CMPNN the type of the edge is estimated by a neural work h_e, and the node-wise convolution kernel are composed by h_e and a global invariant kenrel $\mathbf{k}$. The aggregator $\mathfrak{A}$ can be $\max$ or other set functions.

Then given the ground truth permutation $\hat{\pi}$, the loss becomes

$$\begin{aligned} \ell &= -\sum_i \ln p(\pi(i) = \hat{\pi}(i)) \\ &= \sum_i \left[\ln \sum_j \exp S_1(\mathsf{f}_j, \mathsf{g}_i) - S_1(\mathsf{f}_{\hat{\pi}(i)}, \mathsf{g}_i) \right]. \end{aligned} \tag{8}$$

3.2. Network Architecture

In this section, we give the details of the proposed network. Our neural network consists of several layers of Compositional Message Passing Neural Networks (CMPNN). First of all, we will organize the input feature points as a graph. In our implementation, the graphs are constructed by connecting the k-nearest neighbors with edges. Then the graph and features will be passed to a multilayer CMPNN to get the final features.

Compositional Message Passing Neural Networks Here we propose the CMPNN to get an accelerated version of MPNN. A good properties of CNN is that it comes with a shift invariant global kernel. Our main motivation is to port such a properties to MPNN. Our key observation is that CNN can be view as a graph neural network on a grid graph, and each edge in the grid is associated with a one hot feature vector e_{ij} which indicates the type of the edge. For example, in a convolution layer with 3×3 kernel (shown in the left of Figure 2), there are 9 different types of edges. Assume that we have a 2D convolutional layer which maps a d_{in}-channel image to a d_{out}-channel image with a $m \times m$ kernel, Then the convolutional kernel can be represented as a $m^2 \times d_{\text{out}} \times d_{\text{in}}$ tensor $\mathbf{k}$, and the convolution operation

can be rewrite as [2]

$$\mathbf{y}_i = \sum_{j \in N(i)} \mathbf{e}_{ij}\, \mathbf{k}\, \mathbf{x}_j\,. \tag{9}$$

For a general graph, due to the irregular structure, it is difficult to associate each edge with fixed type. Thus, instead of associating the type for each edge manually, we use a neural network $h_e(\cdot, \boldsymbol{\theta}_e)$ to predict the type of the edges (see Figure 2). Then by introducing a globally invariant kernel $\mathbf{k} \in \mathbb{R}^{t_e \times d_{\text{out}} \times d_{\text{in}}}$, the MPNN (2) can be reformulated as

$$\mathbf{y}_i = \mathop{\mathfrak{A}}_{j \in N(i)} h_e(\mathbf{e}_{ij}, \boldsymbol{\theta}_e)\, \mathbf{k}\, \mathbf{x}_j, \tag{10}$$

where the node-wise convolution kernel is composed of an edge type classifier and a global invariant kernel $\mathbf{k}$.

The above operation can be further decomposed as

$$\hat{\mathbf{x}}_i = \mathbf{k}\, \mathbf{x}_i, \quad \mathbf{y}_i = \mathop{\mathfrak{A}}_{j \in N(i)} h_e(\mathbf{e}_{ij}, \boldsymbol{\theta}_e) \hat{\mathbf{x}}_j, \tag{11}$$

where the first step can be done by matrix multiplication, and the second step can be done by batch matrix multiplication and pooling. As both the operation are implemented efficiently in deep learning packages such as PyTorch[22], the proposed CMPNN becomes quite easy to implement.

Extended CMPNN The original MPNN (2) can be extended as follows[5]

$$\mathbf{k}_{ij} = h_{\mathbf{k}}(\mathbf{e}\,|\,\boldsymbol{\theta}_{\mathbf{k}}) \in \mathbb{R}^{d_{\text{out}} \times 2d_{\text{in}}}, \tag{12}$$
$$\mathbf{y}_i = \mathop{\mathfrak{A}}_{j \in N(i)} \mathbf{k}_{ij}[\mathbf{x}_i, \mathbf{x}_j], \text{ or } \mathbf{y}_i = \mathop{\mathfrak{A}}_{j \in N(i)} \mathbf{k}_{ij}[\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i]$$

to make the learning easier. This extension can also be handled by our CMPNN. Instead of introducing one global $\mathbf{k}$, by introducing two different global invariant kernel $\mathbf{k}_{\text{orig}}$ and $\mathbf{k}_{\text{neighbor}}$, the CMPNN (11) can be extended as follows,

$$\bar{\mathbf{x}}_i = \mathbf{k}_{\text{orig}}\, \mathbf{x}_i, \hat{\mathbf{x}}_i = \mathbf{k}_{\text{neighbor}}\, \mathbf{x}_i, \forall i \in \{1, 2, \ldots, n\}, \tag{13a}$$
$$\mathbf{y}_i = \mathop{\mathfrak{A}}_{j \in N(i)} h_e(\mathbf{e}_{ij}, \boldsymbol{\theta}_e)\big[\bar{\mathbf{x}}_i + \hat{\mathbf{x}}_j\big] \text{ or} \tag{13b}$$
$$\mathbf{y}_i = \mathop{\mathfrak{A}}_{j \in N(i)} h_e(\mathbf{e}_{ij}, \boldsymbol{\theta}_e)\big[\bar{\mathbf{x}}_i + \hat{\mathbf{x}}_j - \hat{\mathbf{x}}_i\big]. \tag{13c}$$

Residual CMPNN He *et al*. [8] proposed the residual network, which helps to train very deep neural works and led to significant performance improvement in various areas including classification, segmentation *etc*. In our network, we also apply the residual link to make the training easier. In normal CNN, a residual block consists of one 1×1 convolution layer, one $m \times m$ convolutional layer and another 1×1 convolution layer. While for GNN, equivalently the 1×1 convolution layer are replaced by fully connected layer, and the $m \times m$ convolution layer is replaced by a CMPNN layer. Our residual block for CMPNN is shown in Figure 3.

[2] We slightly abuse the notation of matrix multiplication for tensors. A $k \times m \times n$ tensor times a n-d vector will be a $k \times m$ matrix. Similarly, a k-d vector times a $k \times m \times n$ tensor will be a $m \times n$ matrix.

Global Pooling In PointNet and its following works [23], global pooling are widely used to help to model to capture more global information to improve the performance of classification and segmentation information. For the feature matching problem, global transformation such as rotation can not be captured by local features easily. As a result, we are also using global pooling to help the network to improve its performance under global deformation.

Implementation Details With the proposed CMPNN and residual CMPNN module, we propose our geometric feature net shown in the right of Figure 3. First we build a directed graph by using k-nearest neighbor from the input feature point. In our implementation, k is set to 8. Then each edge vector can be computed by subtracting the source point from the sink point, and each edge vector is passed to a edge type network, which is a 2-layer MLP. In our implementation, we consider 16 different types of edges (*i.e.* $t_e = 16$). For all CMPNN layers, we are using the extension shown in (13c), and we use maximization as the aggregation function $\mathfrak{A}$. In our network, all MLP layers and CMPNN layers except for the output layer of the edge-type network and the geometric feature network, are followed with a batch-normalization layer [9] and a RELU activation function [20]. In the output layer of the edge-type network, we use softmax as the activation function, and the output of the geometric feature network is normalized to a unit vector. Thus we can use inner product to measure the similarity between the features. Overall our feature matching pipeline is shown on the right of Figure 3. The feature point $\mathcal{F}$ and $\mathcal{G}$ is input to the same geometric feature network to get two new sets of local features $\mathfrak{F}$ and $\mathfrak{G}$. We then use the inner product to generate the similarity matrix. Finally, the Hungarian/Munkres algorithm is used to find the correspondence which maximizes the feature similarity.

3.3. Relationship with Existing Methods

In this section, we will discuss the relationship of our CMPNN with existing methods.

Message Passing Neural Networks Here we show that CMPNN is equivalent to MPNN. Without loss of generality, we assume that the last layer in the kernel generator of MPNN is a linear layer without any activation function[3]. Then the kernel generator $h_{\mathbf{k}}$ can be reformulated as

$$\mathbf{k}_{ij} = \hat{h}_{\mathbf{k}}(\mathbf{e}_{ij}\,|\hat{\boldsymbol{\theta}}_{\mathbf{k}})\, \mathbf{w}_{\mathbf{k}}, \tag{14}$$

where $\hat{h}_{\mathbf{k}}(\mathbf{e}_{ij}\,|\hat{\boldsymbol{\theta}}_{\mathbf{k}})$ is a neural network with d-dimensional output, and $\mathbf{w}_{\mathbf{k}}$ is a $d \times d_{\text{out}} \times d_{\text{in}}$ tensor. Then the MPNN

[3] If the last layer is non-linear, we can simply append a linear identical mapping layer.

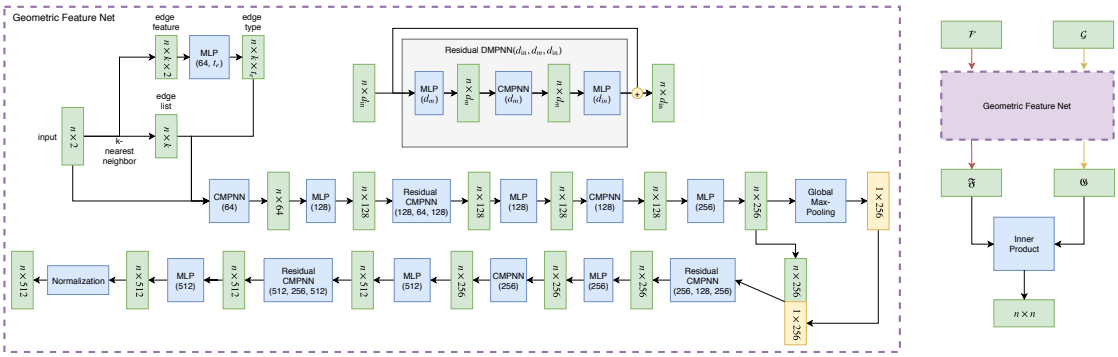


Figure 3: The network architecture used in our network. **Left:** The geometric feature net consists of multiple CMPNN layers. **Right:** The Siamese net framework for the feature matching problem.

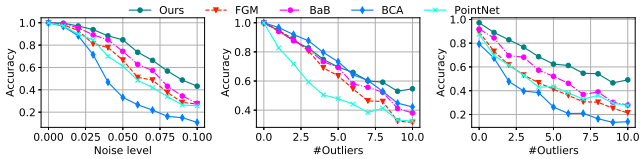


Figure 4: Comparison of feature matching for synthetic datasets. **Left:** Accuracy with different noise level. **Middle:** Accuracy with different number of outliers with noise level 0. **Right:** Accuracy with different number of outliers with noise level 0.025.

(2) can be reformulated as

$$\mathbf{y}_i = \underset{j \in N(i)}{\mathfrak{A}} \hat{h}_{\mathbf{k}}(\mathbf{e}_{ij} \,|\hat{\boldsymbol{\theta}}_{\mathbf{k}})\, \mathbf{w}_{\mathbf{k}}\, \mathbf{x}_j, \tag{15}$$

which has exactly the same formulation as our CMPNN.

PointNet One of the kernel part, the global pooling, in PointNet [23] can be viewed as a special type of Message Passing Neural Network. Assume that for some particular i, we let $N(i) = \{1, 2, \ldots, n\}$ and we let all other $N(i') = \emptyset$. Under this assumption, we let the aggregation operator to be maximization operator. Viewed in that way, the Point Pair Feature Net (PPFNet) [3] can also be viewed as a two layer Message Passing Neural Network with a specific graph structure. As a result, the proposed method can be view as a generalization of PointNet and PPFNet from a specific graph structure to more general graph structures.

4. Experiments

In this section, we compare the proposed methods with existing feature matching methods on the geometric feature matching problem, where only the coordinate of the points are available. In this scenario, the feature matching algorithm must be good enough to capture the geometric and topological relation between feature points. For traditional methods, we compare against two slow but accurate feature matching method that are based on pairwise features:

- Branch and Bound (BaB): Zhang *et al*. [30] proposed a Branch and Bound based method, in which a Lagrangian relaxation based solver is used in a Branch and Bound framework.
- Factorized Graph Matching (FGM) [31]: the factorized graph matching algorithm is based on Convex-Concave Relaxation Procedure. It is one of the most accurate feature matching algorithm that has been proposed so far.

We also compare with a third-order feature based method:

- Block Coordinate Ascent Graph Matching (BCAGM): Ngok *et al*. [21] proposed an efficient higher order assignment solver based on multi-linear relaxation and block coordinate ascent. The algorithm outperforms various higher order matching methods.

For deep learning based methods, as far as the authors know, there is no existing work on the matching problem that uses only the coordinates of feature points. Thus we use the most representative work, PointNet [23], as a baseline. Here we use the same network structure as used in the PointNet for semantic segmentation, but with the last softmax layer replaced by a ℓ_2 normalization layer. We also slightly adapt the code to make PointNet work for 2d feature points. We adopt the same Siamese network structure for PointNet as shown in Figure 3.

Training Protocol The training data available for the feature matching problem using only the coordinate is quite limited. In existing annotated datasets such as Pascal-PF [7], only thousands of matching pairs are provided, where each pair have no more than 20 feature points. Thus in our experiments, we train our model and the comparison methods on the same randomly generated synthetic training set. To do that, we generated 9 million matching pairs synthetically. For each pair, we first randomly generate 30-60 2d reference feature points uniformly from $[-1, 1]^2$, then do a random rotation and add Gaussian noise from $\mathcal{N}(0, 0.05^2)$

to the reference feature points in order to generate the target feature points. Finally, we randomly add 0-20 outliers from $[-1.5, +1.5]^2$ to each pair. In our training, we use the Adam [11] method with learning rate 10^{-3} to optimize all methods, and each method is run on one epoch on the generated data; the algorithms converges after only 0.5 epoch[4].

4.1. Synthetic Dataset

We perform a comparative evaluation of the traditional methods and the two deep learning based methods on random generated feature point sets. In each trial, we generated two identical feature point set $\mathcal{F}$, $\mathcal{G}$ with 20 inliers from $[0, 1]^2$ and later we will add n_{out} outliers to each feature point sets. Then feature points in the second set will then be perturbed by an additive Gaussian noise from $\mathcal{N}(0, \sigma^2)$. For the traditional second-order methods, the feature points are connected by Delaunay triangulation and the length of each edge serves as the edge feature; the edge feature similarity is computed as $s(e, e') = \exp(-\frac{(e1-e2)^2}{0.15})$. For the third order methods, we use the three angles of the triangles from Delaunay triangulation as features, and the similarity function is defined as in [21]. The node similarities of traditional methods are set to 0. For the deep learning methods, the feature points from $\mathcal{F}$ are first normalized as follows

$$\mathcal{F}' = \left\{ f_i' \middle| f_i' = \frac{f_i - \frac{1}{|\mathcal{F}|}\sum_{f\in\mathcal{F}} f}{\|f_i - \frac{1}{|\mathcal{F}|}\sum_{f\in\mathcal{F}} f\|_2}, f_i \in \mathcal{F} \right\}, \quad (16)$$

and the feature points from $\mathcal{G}$ are also normalized in the same way to $\mathcal{G}'$. Then the normalized feature points are passed to the deep learning based methods.

The experiment tested the performance of the methods under three different scenarios; 100 different instances are generated in each parameter setting of each scenario to get the average matching accuracy. In the first scenario (left of Figure 4), the number of outliers n_{out} is set to 0 and we increase the noise level σ^2 form 0 to 0.1 with step-length 0.01. In the second scenario (middle of Figure 4), we set the noise level $\sigma^2 = 0$, and we increase the number of outliers from 0 to 10 with step-length 1. In the third setting, we $\sigma^2 = 0.025$, we increase the number of outliers from 0 to 10 with step-length 1. Under various different settings, it can be observed that the proposed methods outperforms the others in most cases. The traditional methods are particularly sensitive to noise on feature points, while our methods is more robust. When there is no noise and the number of outlier is less than 5, the BCAGM method get slight better results than others; the FGM and BaB method get comparable accuracy with the proposed method, while the proposed method generalized better with more outliers. The PointNet becomes the worst in most cases, this might be caused by its poor ability in propagating local information.

[4]The code is available at https://github.com/zzhang1987/Deep-Graphical-Feature-Learning.

4.2. CMU House Dataset

The CMU house image sequence is a common dataset to test the performance of feature matching algorithms [13, 28, 12, 31, 30, 25]. The data consists of 111 frames of a toy house, and each of the image has been manually labeled with 30 landmarks. For traditional second order method, we use the settings in [31] to construct the matching model, and for BCAGM, we use the same feature and similarity function as Section 4.1. For deep learning model based methods, we normalize the coordinate of the 30 landmarks as in (16). The performance of different methods are tested on all possible image pairs, with separation 10:10:100 frames. Figure 6 show an instance of a matching pair of two frames.

We tested the accuracy and running time on the dataset. In terms of accuracy, all methods except the PointNet get perfect matching when the separation is small. For running time, the Branch and Bound methods can achieve similar speed with the our method when the separation is small.

It is notable that when the separation is small, the matching pairs are quite similar to the matching pairs from our training set, where feature points in one frame can be approximately transformed to another frame with rotation and shift. When the separation is very large (see the example in Figure 6, such transformations do not exists due to large view changes; this is quite far away from the cases in our training set. As our algorithm is able to extract features from local geometric structures, it generalizes to the large view change cases. For runtime with large separation where the view angle changes a lot, our methods is hundreds of times faster than FGM and BaB.

4.3. Real Image Matching

In this experiment, we use the PF-Pascal dataset [7] to evaluate the performance of feature matching algorithms on key-points matching on real images. The PF-Pascal dataset consists of 20 classes of 1351 image pairs. In each image pair, there are 6-18 manually annotated ground-truth correspondences. Here, for traditional methods, the feature similarity model is the same as in the CMU house dataset. For deep learning based methods, we also follow the same protocol as in the CMU house dataset. Typical images pairs and matching results are shown in Figure 5. In this dataset, we consider two different settings. In the first setting, we directly use the coordinate of feature points as input for all algorithms. In the second settings, we first apply a random rotation to the feature points in the first points, and use the rotated feature points as input. The matching accuracy are shown in Table 1. When no random rotation is applied, the proposed method outperforms all other algorithms, and the PointNet is the second best. When random rotations are

Table 1: Matching Accuracy On Pascal-PF Dataset. **Top four rows:** Results on original feature points. **Bottom four rows:** Results on random rotated feature points

	aero-plane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	dining-table	dog	horse	motor-bike	person	potted-plant	sheep	sofa	train	tv-monitor	average
#pairs	69	133	50	28	42	140	84	119	59	15	38	106	39	120	56	35	6	59	88	65	
BaB[30]	70.0	80.5	73.7	74.6	37.3	64.4	81.2	60.6	68.3	63.2	52.5	50.5	67.5	74.3	52.3	38.8	26.7	73.1	86.8	29.8	61.3
FGM[31]	57.4	73.3	67.6	71.0	40.9	60.1	74.0	54.2	63.9	51.7	52.5	48.1	62.0	70.5	50.1	49.2	36.2	65.9	87.5	29.0	58.3
BCAGM[21]	60.0	62.8	59.6	66.3	32.7	58.2	70.6	61.7	62.6	52.3	52.9	45.2	48.4	51.7	40.5	48.2	29.5	73.1	87.0	37.2	55.0
PointNet[23]	54.8	70.2	65.2	73.7	85.3	90.0	73.4	63.4	55.2	78.4	78.4	52.5	58.0	64.2	57.4	68.9	50.5	74.0	88.1	91.9	69.7
Ours	**76.1**	**89.8**	**93.4**	**96.4**	**96.2**	**97.1**	**94.6**	**82.8**	**89.3**	**96.7**	**89.7**	**79.5**	**82.6**	**83.5**	**72.8**	**76.7**	**77.1**	**97.3**	**98.2**	**99.5**	**88.5**
BaB[30]	69.7	80.2	73.7	**74.6**	37.3	**64.4**	81.2	60.4	68.3	63.2	52.5	51.0	67.2	74.2	53.7	38.8	26.7	73.1	86.8	29.8	61.3
FGM[31]	59.4	70.8	68.9	67.5	38.6	61.6	76.4	48.8	61.8	49.7	**53.2**	45.4	67.6	71.7	49.9	43.5	27.6	67.9	85.0	28.9	57.2
BCAGM[21]	60.0	62.9	59.6	66.3	32.7	58.2	70.4	61.8	62.6	52.3	52.9	45.1	48.4	50.9	41.1	**48.2**	29.5	73.1	**87.0**	**37.2**	55.0
PointNet[23]	35.3	34.2	57.0	50.1	38.7	31.7	45.4	45.8	32.7	64.3	45.5	36.9	45.3	45.8	22.2	38.8	49.0	28.8	54.1	28.9	41.5
Ours	**74.5**	**88.5**	**89.6**	70.8	**85.7**	53.6	**87.2**	**66.8**	**77.9**	**89.3**	45.9	**65.4**	**79.6**	**81.4**	**75.2**	36.7	**52.9**	**83.3**	68.9	26.6	**69.9**

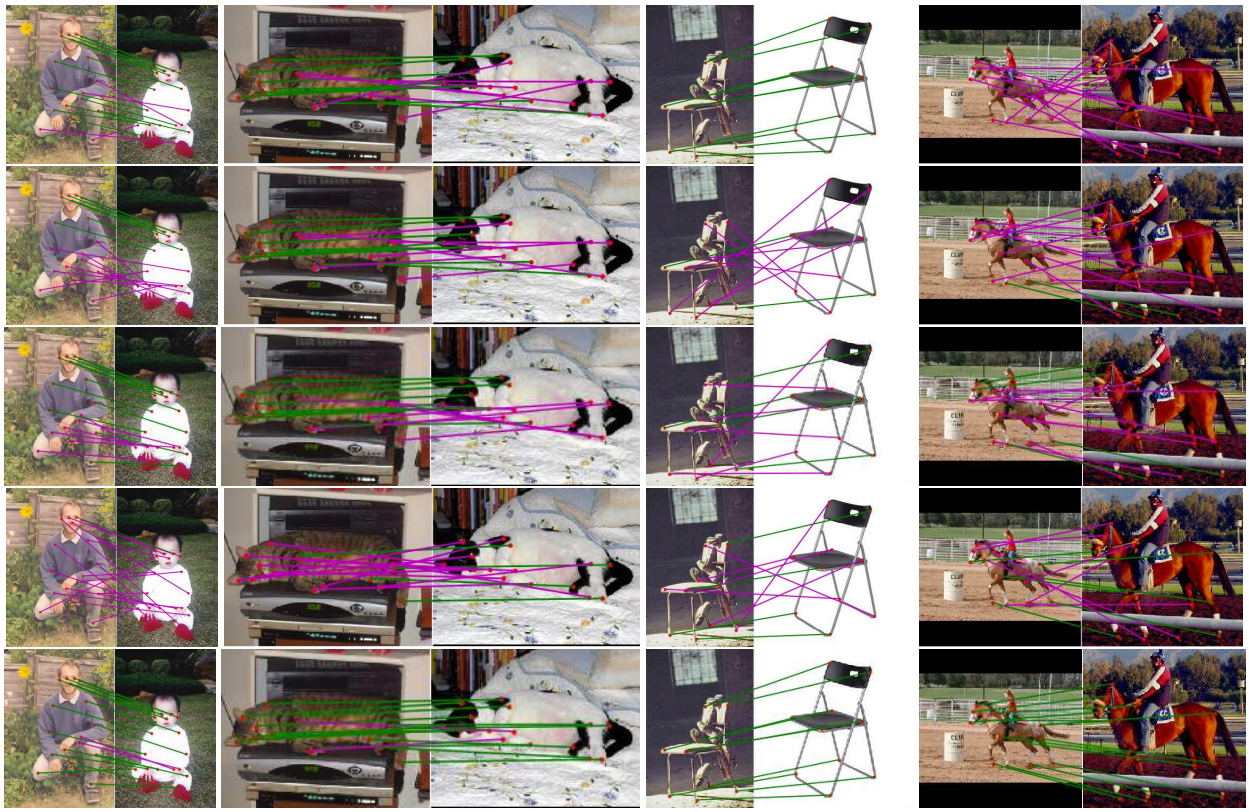


Figure 5: Key-points Matching Results on Pascal-PF dataset. From top to down the methods are BaB, FGM, BCAGM, PointNet and Ours. The correct matchings are in green and the wrong matchings are in purple.

applied, the proposed methods outperforms the other in 14 classes, and its over-all accuracy also outperforms the others. However, the accuracy of the PointNet drops to being the worst, which suggests that the PointNet does not generalize well for rotation.

4.4. Architecture Design Analysis

In this section, we present the ablation study results, particularly on the effects of the hyper parameters of our module. We train different network using the same protocol, and

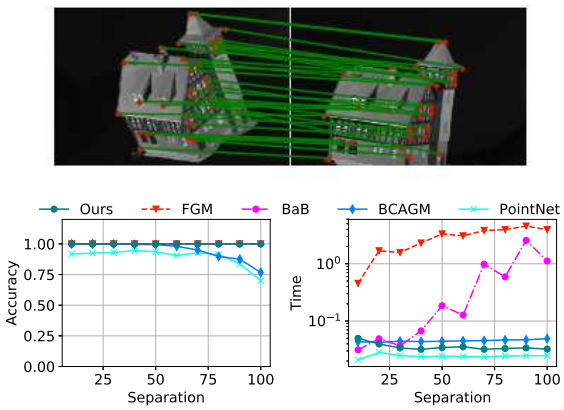


Figure 6: Performance comparison of different feature matching methods on CMU house data. **Left:** Our typical matching result. **Right:** Accuracy and running time v.s. separation.

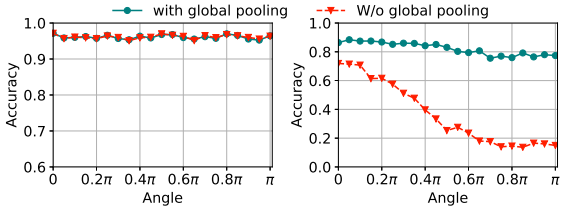
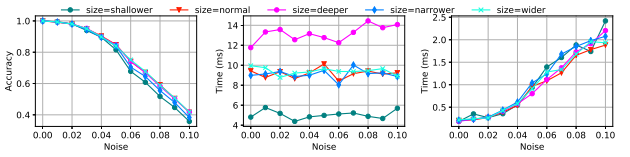


Figure 7: Accuracy comparison for network with/without global pooling. **Left:** No outlier, noise-level $\sigma^2 = 0.025$; **Right:** $n_{\text{out}} = 2$, noise-level $\sigma^2 = 0$.

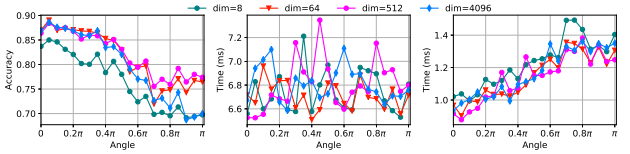
test the network with random generated matching pairs.

The global pooling As mentioned in Section 3.2, we use the CMPNN to get rich local features and the global pooling layer is used to provide global features. Here we trained a network with the global pooling layer removed, and compare its performance to the complete model. Intuitively, the global pooling layer may help the network to be more robust under global deformation (*e.g.* rotation). Thus we analyzed the performance of the two network with respect to different rotations using the same random feature generating scheme as Section 4.1. Here we consider two different scenarios. In the first scenario, Gaussian noise is added to the feature points and in the second scenario we add outliers to the feature points. The typical result for the two scenarios is shown in Figure 7. When there are no outliers, there is no significant performance difference between model, and when there are outliers, global pooling can significantly improve the performance of the model.

Network depth & width We compared the accuracy, inference time and inference memory usage (batch size=1) of different models. The inference time are broken down into the neural network inference time and the linear assignment problem (LAP) inference time. By removing the first residual CMPNN layer and the second CMPNN layer in our model, we get a shallower model; by repeating the second CMPNN layer three times we get a deeper model.



(a) Results of network with different size; σ^2=0.05, #outliers=0. GPU Mem(MB): shallower:59; normal:76; deeper:103; narrower:20; wider:297.



(b) Results of different feature dimension; σ^2=0, #outliers=2. GPU Mem(MB): dim=8:72; dim=64:73; dim=512:76; dim=4096:104.

Figure 8: Performance comparison of different network models. From Left to Right: accuracy, neural network inference time and LAP inference time.

We shrink/enlarge the dimension of intermediate features by a factor of 2 to get a narrower/wider model. Figure 8 (a) show that the shallower and narrower model have worse performance with high noise level. The deeper and wider model have similar performance as the normal one, but uses longer inference time and more memory.

We also evaluated different dimensions of output features from 8 to 4096 by modifying the number of output units in the last layer (Results in Figure 8 (b)). In terms of running time, there is no significant difference between different settings. Meanwhile, the accuracy of different setting varies when there is outlier and rotation, the setting dim=8 has significant lower accuracy than the others, and the setting dim=4096 has lower accuracy than dim=64 and dim=512. The setting dim=512 is slight better than dim=64.

5. Conclusion

In this paper, we proposed a graph neural network which can transform weak local geometric features into rich local features for the geometric feature matching problem. The proposed model is trained with synthetic data, and is then applied to both synthetic and real-world feature matching problems with feature deformations caused by noise, rotation and outliers. The experimental results suggest that the local features produced by our graph neural networks can be used with simple inference algorithms to outperform traditional methods that use handcrafted inference algorithms. The results suggest that with quite limited local information, the graph neural network is still able to get rich local representations through message passing between nodes.

Acknowledgements

This research is supported by the National Research Foundation Singapore under its AI Singapore Programme grant AISG-RP-2018-006. This work is also partly supported by ARC discovery project DP160100703.

References

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006. 1

[2] Rainer E Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment problems*. Springer, 2009. 1

[3] Haowen Deng, Tolga Birdal, and Slobodan Ilic. PPFNet: Global context aware local features for robust 3d point matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 195–205, 2018. 2, 5

[4] Olivier Duchenne, Armand Joulin, and Jean Ponce. A graph-matching kernel for object categorization. In *2011 International Conference on Computer Vision*, pages 1792–1799. IEEE, 2011. 1

[5] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017. 2, 4

[6] Michelle Guo, Edward Chou, De-An Huang, Shuran Song, Serena Yeung, and Li Fei-Fei. Neural graph matching networks for fewshot 3d action recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 653–669, 2018. 1

[7] Bumsub Ham, Minsu Cho, Cordelia Schmid, and Jean Ponce. Proposal flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3475–3484, 2016. 1, 5, 6

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015. 4

[10] U. Iqbal, A. Milan, and J. Gall. PoseTrack: Joint multi-person pose estimation and tracking. In *CVPR*, 2017. 1

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[12] Jungmin Lee, Minsu Cho, and Kyoung Mu Lee. Hyper-graph matching via reweighted random walks. In *CVPR 2011*, pages 1633–1640. IEEE, 2011. 1, 2, 6

[13] Marius Leordeanu, Martial Hebert, and Rahul Sukthankar. An integer projected fixed point method for graph matching and map inference. In *Advances in neural information processing systems*, pages 1114–1122, 2009. 1, 2, 6

[14] Hongsheng Li, Xiaolei Huang, and Lei He. Object matching using a locally affine invariant and linear programming techniques. *IEEE transactions on pattern analysis and machine intelligence*, 35(2):411–424, 2013. 1, 2

[15] Zhi-Yong Liu, Hong Qiao, Xu Yang, and Steven CH Hoi. Graph matching by simplified convex-concave relaxation procedure. *International Journal of Computer Vision*, 109(3):169–186, 2014. 1

[16] David G Lowe et al. Object recognition from local scale-invariant features. In *iccv*, volume 99, pages 1150–1157, 1999. 1

[17] João Maciel and João P Costeira. A global solution to sparse correspondence problems. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (2):187–199, 2003. 1

[18] Anton Milan, S Hamid Rezatofighi, Ravi Garg, Anthony Dick, and Ian Reid. Data-driven approximations to np-hard problems. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 2

[19] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957. 1

[20] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010. 4

[21] Quynh Nguyen, Antoine Gautier, and Matthias Hein. A flexible tensor block coordinate ascent scheme for hypergraph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5270–5278, 2015. 5, 6, 7

[22] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 4

[23] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. 2, 4, 5, 7

[24] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. 2011. 1

[25] Paul Swoboda, Carsten Rother, Hassan Abu Alhaija, Dagmar Kainmuller, and Bogdan Savchynskyy. A study of lagrangean decompositions and dual ascent solvers for graph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1607–1616, 2017. 1, 2, 6

[26] Junchi Yan, Chao Zhang, Hongyuan Zha, Wei Liu, Xiaokang Yang, and Stephen M Chu. Discrete hyper-graph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1520–1528, 2015. 1, 2

[27] Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. Lift: Learned invariant feature transform. In *European Conference on Computer Vision*, pages 467–483. Springer, 2016. 1

[28] Ron Zass and Amnon Shashua. Probabilistic graph and hypergraph matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008. 1, 2, 6

[29] Zhen Zhang, Julian McAuley, Yong Li, Wei Wei, Yanning Zhang, and Qinfeng Shi. Dynamic programming bipartite belief propagation for hyper graph matching. In *IJCAI*, pages 4662–4668, 2017. 1

[30] Zhen Zhang, Qinfeng Shi, Julian McAuley, Wei Wei, Yanning Zhang, and Anton Van Den Hengel. Pairwise matching through max-weight bipartite belief propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1202–1210, 2016. 1, 2, 5, 6, 7

[31] Feng Zhou and Fernando De la Torre. Factorized graph

matching. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 127–134. IEEE, 2012. 1, 2, 5, 6, 7