DeepIM: Deep Iterative Matching for 6D Pose Estimation

Yi Li† Gu Wang† Xiangyang Ji† Yu Xiang‡ Dieter Fox‡

† Tsinghua University
‡ University of Washington and NVIDIA Research
yili.matrix@gmail.com, wangg16@mails.tsinghua.edu.cn,
xyji@tsinghua.edu.cn, {yux, dieterf}@nvidia.com

将对象的渲染图像与输入图像进行匹配可以产生精确的结果

Abstract. Estimating the 6D pose of objects from images is an important problem in various applications such as robot manipulation and virtual reality. While direct regression of images to object poses has limited accuracy, matching rendered images of an object against the input image can produce accurate results. In this work, we propose a novel deep neural network for 6D pose matching named DeepIM. Given an initial pose estimation, our network is able to iteratively refine the pose by matching the rendered image against the observed image. The network is trained to predict a relative pose transformation using an untangled representation of 3D location and 3D orientation and an iterative training process. Experiments on two commonly used benchmarks for 6D pose estimation demonstrate that DeepIM achieves large improvements over state-of-the-art methods. We furthermore show that DeepIM is able to match previously unseen objects.

Keywords: 3D Object Recognition, 6D Object Pose Estimation

1 Introduction

Localizing objects in 3D from images is important in many real world applications. For instance, in a robot manipulation task, the ability to recognize the 6D pose of objects, i.e., 3D location and 3D orientation of objects, provides useful information for grasp and motion planning. In a virtual reality application, 6D object pose estimation enables virtual interactions between humans and objects. While several recent techniques have used depth cameras for object pose estimation, such cameras have limitations with respect to frame rate, field of view, resolution, and depth range, making it very difficult to detect small, thin, transparent, or fast moving objects. Unfortunately, RGB-only 6D object pose estimation is still a challenging problem, since the appearance of objects in the images changes according to a number of factors, such as lighting, pose variations, and occlusions between objects. Furthermore, a robust 6D pose estimation method needs to handle both textured and textureless objects.

Traditionally, the 6D pose estimation problem has been tackled by matching local features extracted from an image to features in a 3D model of the object

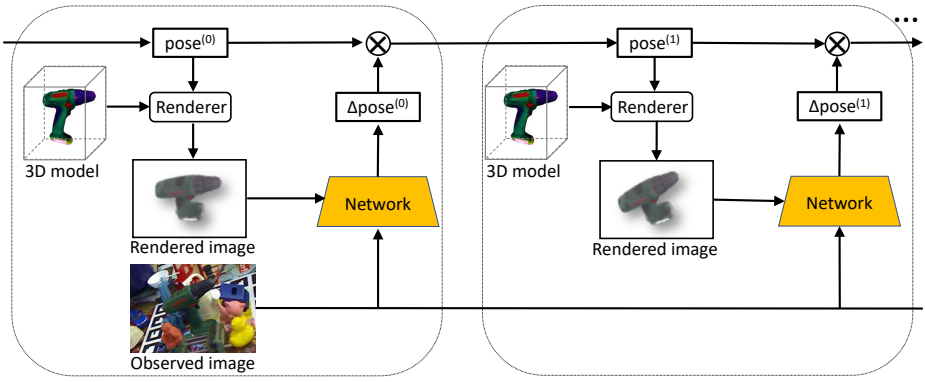


Fig. 1: We propose DeepIM, a deep iterative matching network for 6D object pose estimation. The network is trained to predict a relative SE(3) transformation that can be applied to an initial pose estimation for iterative pose refinement.

to be detected [1,2,3]. By using the 2D-3D correspondences, the 6D pose of the object can be recovered. Unfortunately, such methods cannot handle textureless objects well since only few local features can be extracted for them. To handle textureless objects, two classes of approaches were proposed in the literature. Methods in the first class learn to estimate the 3D model coordinates of pixels or key points of the object in the input image. In this way, the 2D-3D correspondences are established for 6D pose estimation [4,5,6]. Methods in the second class convert the 6D pose estimation problem into a pose classification problem by discretizing the pose space [7] or into a pose regression problem [8]. These methods are able to deal with textureless objects, but they are not able to achieve highly accurate pose estimation, since small errors in the classification or regression stage directly lead to pose mismatches. A common way to improve the pose accuracy is via pose refinement: Given an initial pose estimation, a synthetic RGB image can be rendered and used to match against the target input image. Then a new pose estimation is computed to increase the matching score. Existing methods for pose refinement use either hand-crafted image features [9] or matching score functions [5].

In this work, we propose DeepIM, a new refinement technique based on a deep neural network for iterative 6D pose matching. Given an initial 6D pose estimation of an object in a test image, DeepIM predicts a relative SE(3) transformation that matches a rendered view of the object against the observed image. By iteratively re-rendering the object based on the improved pose estimates, the two input images to the network become more and more similar, thereby enabling the network to generate more and more accurate pose estimates. Fig. 1 illustrates the iterative matching procedure of our network for pose refinement.

This work makes the following main contributions. First, we introduce a deep network for iterative, image-based pose refinement that does not require any hand-crafted image features, automatically learning an internal refinement mechanism. Second, we propose an untangled representation of the SE(3) trans-

formation between object poses to achieve accurate pose estimates. This representation also enables our approach to refine pose estimates of unseen objects. Finally, we have conducted extensive experiments on the LINEMOD [7] and the Occlusion [4] datasets to evaluate the accuracy and various properties of DeepIM. These experiments show that our approach achieves large improvements over state-of-the-art RGB-only methods on both datasets. Furthermore, initial experiments demonstrate that DeepIM is able to accurately match poses for unseen objects when trained on other objects.

The rest of the paper is organized as follows. After reviewing related works in Section 2, we describe our approach for pose matching in Section 3. Experiments are presented in Section 4, and Section 5 concludes the paper.

2 Related work

We review representative relative works on 6D pose estimation in the literature.

RGB-D based 6D Pose Estimation: When depth information is available, it can be combined with RGB images to improve 6D pose estimation. A common strategy of using depth is to convert a depth image into a 3D point cloud, and then match the 3D model of an object against the 3D point cloud. For example, [7] renders the 3D model of an object into templates of surface normals, and then match these templates against normals computed from the point cloud. [4,10,11] regress each pixel on the object in the input image to the 3D coordinate of that pixel on the 3D model. When depth images are available, the 3D coordinate regression establishes correspondences between 3D scene points and 3D model points, from which the 6D pose of the object can be computed by solving a least-squares problem. For pose refinement, the Iterative Closest Point (ICP) algorithm is widely used in the literature to refine an initial pose estimate [7,11,12]. However, ICP is sensitive to the initial estimate and may converge to local minima, especially under occlusions.

RGB based 6D Pose Estimation Traditionally, pose estimation using RGB images is tackled by matching local features [1,2,3]. However, these methods cannot handle textureless objects very well. Hence, several approaches were proposed to apply machine learning, especially deep learning techniques, for 6D pose estimation using RGB images only [4,13]. The state-of-the-art methods in the literature [5,14,6,8] augment deep learning based object detection or segmentation methods [15,16,17,18] for 6D pose estimation. For example, [5,9] utilize deep neural networks to detect key points of the objects, and then compute the 6D pose by solving the PnP problem. [14,8] employ deep neural networks to detect objects in the input image, and then classify or regress the detected object to its pose. Although these methods significantly improve the pose estimation accuracy compared to the traditional methods using RGB images only, their performance is still not comparable to RGB-D based methods. We believe that this performance gap is so large due to the lack of an effective pose refinement

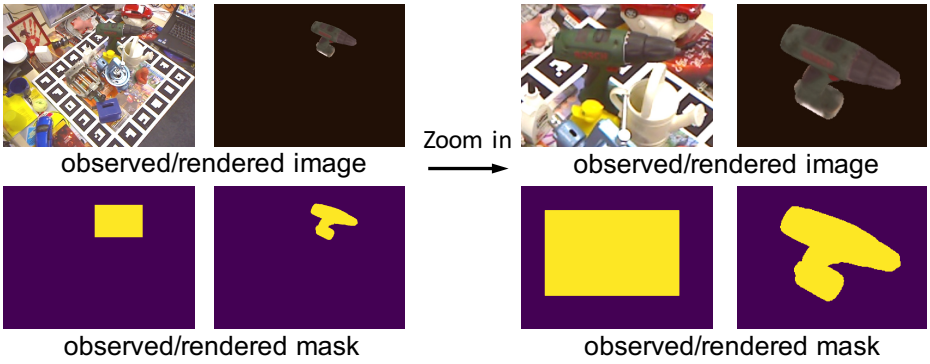


Fig. 2: DeepIM operates on zoomed in, up-sampled versions of the input image, the rendered image, and the two object masks (480×640 in our case after zooming in).

procedure using RGB images only. Our work is complementary to existing 6D pose estimation methods by providing a novel iterative pose matching network for pose refinement on RGB images. The most relevant works to ours are the object pose refinement network in [5] and the iterative hand pose estimation approaches in [19,20]. Compared to these techniques, our network is designed to directly regress to the relative SE(3) transformation, rather than estimating position corrections to individual bounding box corners or key points. We are able to do this due to our untangled representation of rotation and translation and the reference frame we used for rotation, which also allows our approach to match unseen objects. As discussed by [21] in the context of 3D bounding box detection, the choice of reference frame is important to achieve good pose estimation results.

3 DeepIM Framework

In this section, we describe our deep iterative matching network for 6D pose estimation. Given an observed image and an initial pose estimate of an object in the image, we design the network to directly output a relative SE(3) transformation that can be applied to the initial pose to improve the estimate. We first present our strategy of zooming in the observed image and the rendered image that are used as inputs of the network. Then we describe our network architecture for pose matching. After that, we introduce an untangled representation of the relative SE(3) transformation and a new loss function for pose regression. Finally, we describe our procedure for training and testing the network.

3.1 High-resolution Zoom In

It can be difficult to extract useful features for matching if objects in the input image are very small. In order to obtain enough details for pose matching, we zoom in the observed image and the rendered image before feeding them into

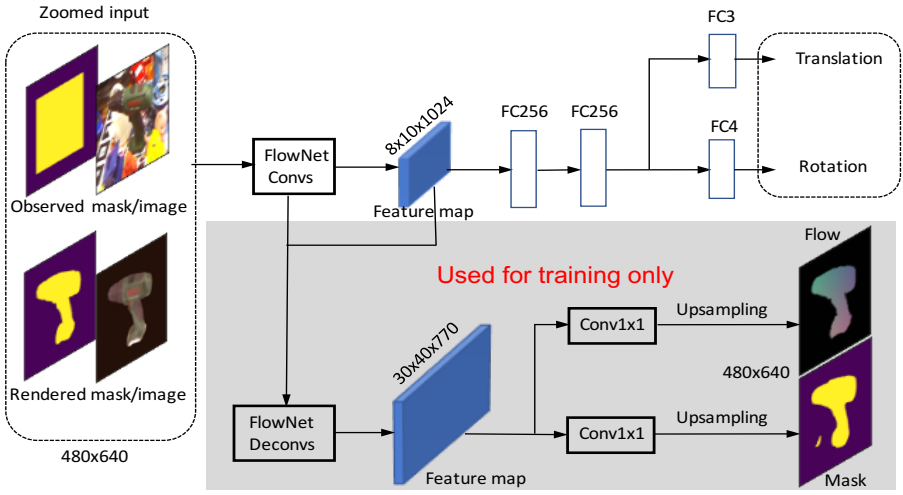


Fig. 3: DeepIM uses a FlowNetSimple backbone to predict a relative SE(3) transformation to match the observed and rendered image of an object. Additional mask and flow losses improve stability during training.

the network, as shown in Fig. 2. Specifically, in the i-th stage of the iterative matching, given a 6D pose estimate $\mathbf{p}^{(i-1)}$ from the previous step, we render a synthetic image using the 3D object model viewed according to $\mathbf{p}^{(i-1)}$. We additionally generate one foreground mask for the observed image and rendered image. The four images are cropped using an enlarged bounding box according to the observed mask and the rendered mask, where we make sure the enlarged bounding box has the same aspect ratio as the input image and is centered at the 2D projection of the origin of the 3D object model. Finally, we zoom in and perform bilinear up-sampling to achieve the same size as the original image (480×640 in our experiments). Importantly, the aspect ratio of the object is not changed during this operation (see Appendix A for more details).

3.2 Network Structure

Fig. 3 illustrates the network architecture of DeepIM. The observed image, the rendered image, and the two masks, are concatenated into an eight-channel tensor that is inputted to the network (3 channels for observed/rendered image, 1 channel for each mask). We use the FlowNetSimple architecture from [22] as the backbone of our network, which is trained to predict optical flow between a pair of images. We have also tried using the VGG16 image classification network [23] as the backbone network, but the results were very poor, confirming the intuition that a representation related to estimating optical flow is very useful for pose matching. The pose estimation branch takes the feature map after 11 convolution layers from FlowNetSimple as input. It contains two fully-connected layers each with dimension 256, followed by two additional fully-connected lay-

ers for predicting the quaternion of the 3D rotation and the 3D translation, respectively. During training, we also add two auxiliary branches to regularize the feature representation of the network and increase training stability. One branch is trained for predicting optical flow between the rendered image and the observed image, and the other branch for predicting the foreground mask of the object in the observed image.

3.3 Untangled Transformation Representation

The representation of the relative SE(3) transformation $\mathbf{\Delta p}$ between the current object pose estimate and the target object pose has important ramifications for the performance of the deep network. In a nutshell, we want the following properties from this representation. First, it should be independent of the specific local coordinate frame specified for a 3D object model. Otherwise, the network would have to learn this coordinate frame for every object, potentially reducing its performance and making it inapplicable to unknown objects. Second, the pose changes predicted by the network should be as closely as possible related to consistent operations in the input images. We achieve this by choosing coordinate frames such that 3D object translations correspond to pixel-wise translations in the image plane along with a change in scale, and 3D rotations correspond to rotations around the image center along with a pan and tilt of the image. Using such a representation, the DeepIM network can operate independently of the actual size of the object and its internal model coordinate framework. It only has to learn to transform the rendered image such that it becomes more similar to the observed image.

To see, consider we represent the object pose and transformation in the camera frame of reference, using a quaternion or Euler angle for the rotation matrix $\mathbf{R}_{\Delta}$ and a vector for the translation $\mathbf{t}_{\Delta}$, so that the full transformation matrix can be written as $[\mathbf{R}_{\mathbf{\Delta}}|\mathbf{t}_{\Delta}]$. Given an source object pose $[\mathbf{R}_{\text{src}}|\mathbf{t}_{\text{src}}]$, the transformed target pose would be as follows:

$$\begin{aligned} \mathbf{R}_{\text{tgt}} &= \mathbf{R}_{\Delta}\mathbf{R}_{\text{src}} \\ \mathbf{t}_{\text{tgt}} &= \mathbf{R}_{\Delta}\mathbf{t}_{\text{src}} + \mathbf{t}_{\Delta}, \end{aligned} \tag{1}$$

where $[\mathbf{R}_{\text{tgt}}|\mathbf{t}_{\text{tgt}}]$ denotes the target pose. The $\mathbf{R}_{\Delta}\mathbf{t}_{\text{src}}$ term in computing $\mathbf{t}_{\text{tgt}}$ indicates that a rotation will cause the object not only to rotate, but also translate in the image even if the translation vector $\mathbf{t}_{\Delta}$ equals to zero. Furthermore, the translation $\mathbf{t}_{\Delta}$ is given in the metric of the 3D space (meter, for instance), which couples object size with distance in the metric space, thereby requiring the network to memorize the size of each object if it has to translate a mismatch in images to distance offset. It is obvious that such a representation is not appropriate.

To eliminate such problems, we propose to decouple the estimation of the relative rotation $\mathbf{R}_{\Delta}$ and the relative translation $\mathbf{t}_{\Delta}$. For rotation, we move the center of rotation from the origin of the camera frame to the center of the object in the camera frame, given by the current pose estimate. In this way, a

rotation transformation would not change the translation of the object in the camera frame. The remaining question is how to choose the axes of the coordinate frame for rotation. One possible way is to use the axes of the coordinate frame as specified in the 3D object model. However, such a representation would require the network to memorize the coordinate frames of each object and recognize them during inference, which makes the training more difficult and cannot be generalized to pose matching of unseen objects. Instead, we use axes parallel to the axes of the camera coordinate frame when computing the relative rotation. By doing so, the network can be trained to estimate the relative rotation independently of the coordinate frame of the 3D object model. Our experiments in Sec. 4.2 also show that such a representation produces far better results than using the axes of the coordinate frame of the 3D object model.

To estimate the relative translation, let $\mathbf{t}_{\text{tgt}} = (x_{\text{tgt}}, y_{\text{tgt}}, z_{\text{tgt}})$ and $\mathbf{t}_{\text{src}} = (x_{\text{src}}, y_{\text{src}}, z_{\text{src}})$ be the target translation and the source translation. Then a straightforward way to represent $\mathbf{t}_\Delta = (\Delta_x, \Delta_y, \Delta_z)$ is

$$\begin{aligned} \Delta_x &= x_{\text{tgt}} - x_{\text{src}}, \\ \Delta_y &= y_{\text{tgt}} - y_{\text{src}}, \\ \Delta_z &= z_{\text{tgt}} - z_{\text{src}}. \end{aligned} \tag{2}$$

However, it is not easy for the network to estimate the relative translation in 3D space given only 2D images due to the missing of depth information. The network has to recognize the size of the object, and map the translation in 2D space to 3D according to the object size. Such a representation is not only difficult for the network to learn, but also has problems when dealing with objects with similar appearance but different sizes or unseen objects.

Instead of training the network to directly regress to the vector in the 3D space, we propose to regress to the object changes in the 2D image space. Specifically, we train the network to regress to the relative translation $\mathbf{t}_\Delta = (v_x, v_y, v_z)$, where v_x and v_y denote the number of pixels the object should move along the image x-axis and y-axis and v_z is the scale change of the object:

$$\begin{aligned} v_x &= f_x(x_{\text{tgt}}/z_{\text{tgt}} - x_{\text{src}}/z_{\text{src}}), \\ v_y &= f_y(y_{\text{tgt}}/z_{\text{tgt}} - y_{\text{src}}/z_{\text{src}}), \\ v_z &= \log(z_{\text{src}}/z_{\text{tgt}}), \end{aligned} \tag{3}$$

where f_x and f_y denote the focal lengths of the camera. The scale change v_z is defined to be independent of the absolute object size or distance by using the ratio between the distances of the rendered and observed object. We use logarithm for v_z to make sure that value zero corresponds to no change in scale or distance. Considering the fact that f_x and f_y are constant for a specific dataset, we simply fix it to 1 in training the network.

Our representation of the relative transformation has several advantages. First, rotation does not influence the estimation of translation, so that the translation no longer needs to offset the movement caused by rotation around the camera center. Second, the intermediate variables v_x, v_y, v_z represent simple

translations and scale change in the image space. Third, this representation does not require any prior knowledge of the object and can adapt to situations where some objects share a similar appearance while having different sizes.

3.4 Matching Loss

A straightforward way to train the pose estimation network is to use separate loss functions for rotation and translation. For example, we can use the angular distance between two rotations to measure the rotation error and use the L1 distance to measure the translation error. However, using two different loss functions for rotation and translation suffers from the difficulty of balancing the two losses. [24] proposed a geometric reprojection error as the loss function for pose regression that computes the average distance between the 2D projections of 3D points in the scene using the ground truth pose and the estimated pose. Considering the fact that we want to accurately predict the object pose in 3D, we introduce a modified version of the geometric reprojection loss in [24], and we call it the Point Matching Loss. Given the ground truth pose $\mathbf{p} = [\mathbf{R}|\mathbf{t}]$ and the estimated pose $\hat{\mathbf{p}} = [\hat{\mathbf{R}}|\hat{\mathbf{t}}]$, the point matching loss is computed as:

$$L_{\text{pose}}(\mathbf{p}, \hat{\mathbf{p}}) = \frac{1}{n} \sum_{j=1}^{n} L_1\big((\mathbf{R}\mathbf{x}_j + \mathbf{t}) - (\hat{\mathbf{R}}\mathbf{x}_j + \hat{\mathbf{t}})\big), \tag{4}$$

where $\mathbf{x}_j$ denotes a randomly selected 3D point on the object model and n is the total number of points used in computing the loss (we choose 3,000 points in our experiments). The point matching loss computes the average L1 distance between 3D points transformed by the ground truth pose and the estimate pose. In this way, it measures how the transformed 3D models match against each other for pose estimation.

3.5 Training and Testing

To train the network, we assume that we have 3D object models and images annotated with ground truth 6D object poses. By adding noises to the ground truth poses as the initial poses, we can generate the required observed and rendered inputs to the network along with the pose target output that is the pose difference between ground truth pose and its noisy version. Then we can train the network to predict the relative transformation between the initial pose and the target pose.

During testing, we find that the iterative pose refinement procedure can significantly improve the pose estimation accuracy. To see why multiple iterations can improve results, let $\mathbf{p}^{(i)}$ be the pose estimate after the i-th iteration of the network. If the initial pose estimate $\mathbf{p}^{(0)}$ is relatively far from the correct pose, the rendered image $\mathbf{x}_{\text{rend}}(\mathbf{p}^{(0)})$ may only have little viewpoint overlap with the observed image $\mathbf{x}_{\text{obs}}$. In such cases, it is very difficult to accurately estimate the relative pose transformation $\mathbf{\Delta p}^{(0)}$ directly. This task is even harder if the

network has no priori knowledge about the object to be matched. In general, it is reasonable to assume that if the network improves the pose estimate $\mathbf{p}^{(i+1)}$ by updating $\mathbf{p}^{(i)}$ with $\mathbf{\Delta p}^{(i)}$ in the i-th iteration, then the image rendered according to this new estimate, $\mathbf{x}_{\text{rend}}(\mathbf{p}^{(i+1)})$ is also more similar to the observed image $\mathbf{x}_{\text{obs}}$ than $\mathbf{x}_{\text{rend}}(\mathbf{p}^{(i)})$ was in the previous iteration, thereby providing input that can be matched more accurately.

However, we found that, if we train the network to regress the relative pose in a single step, the estimates of the trained network do not improve over multiple iterations in testing. To generate a more realistic data distribution for training similar to testing, we perform multiple iterations during training as well. Specifically, for each training image and pose, we apply the transformation predicted from the network to the pose and use the transformed pose estimate as another training example for the network in the next iteration. By repeating this process multiple times, the training data better represents the test distribution and the trained network also achieves significantly better results during iterative testing (such an approach has also proven useful for iterative hand pose matching [20]).

4 Experiments

We conduct extensive experiments on the LINEMOD dataset [7] and the Occlusion LINEMOD dataset [10] to evaluate our DeepIM framework for 6D object pose estimation. We test different properties of DeepIM and show that it surpasses other RGB-only methods by a large margin. We also show that our network can be applied to pose matching of unseen objects during training.

4.1 Implementation Details

Training: We use the pre-trained FlowNetSimple provided by [22] to initialize the weights in our network. Weights of the new layers are randomly initialized, except for the additional weights in the first conv layer that deals with the input masks and the fully-connected layer that predicts the translation, which are initialized with zeros. Other than predicting the pose transformation, the network also predicts the optical flow and the foreground mask in the image. Although including the two additional losses in training does not increase the pose estimation performance, we found that they can help to make the training more stable. Specifically, we use the optical flow loss L_{flow} as in FlowNet [22] and the sigmoid cross-entropy loss as the mask loss L_{mask}.

Two deconvolutional blocks in FlowNet are inherited to produce the feature map used for the mask and the optical flow prediction, whose spatial scale is 0.0625. Two 1×1 convolutional layers with output channel 1 (mask prediction) and 2 (flow prediction) are appended after this feature map. The predictions are then bilinearly up-sampled to the original image size (480×640) to compute losses. The overall loss is

$$L = \alpha L_{\text{pose}} + \beta L_{\text{flow}} + \gamma L_{\text{mask}} \,, \tag{5}$$

where we use $\alpha = 0.1$, $\beta = 0.25$, $\gamma = 0.03$ throughout the experiments (except some of our ablation studies).

Each training batch contains 16 images. We train the network with 4 GPUs where each GPU processes 4 images. We generate 4 items for each image as described in Sec. 3.1: the observed image $\mathbf{x}_{\text{obs}}$, the observed mask $\mathbf{m}_{\text{obs}}$ which shows the tightest bounding box of the ground truth segmentation, the rendered image $\mathbf{x}_{\text{rend}}$ using the initial pose and its corresponding mask $\mathbf{m}_{\text{rend}}$ which is the pixel-level foreground mask. In training, the observed mask is randomly dilated with no more than 10 pixels to avoid over-fitting.

Testing: The mask prediction branch and the optical flow branch are removed during testing. Since there is no ground truth segmentation of the object in testing, we use the tightest bounding box of the rendered mask $\mathbf{m}_{\text{rend}}$ instead, so the network searches the neighborhood near the estimated pose to find the target object to match. Unless specified, we use the pose estimates from PoseCNN [8] as the initial poses. Our DeepIM network runs at 12 fps per object using the NVIDIA 1080 Ti GPU with 2 iterations during testing.

4.2 Experiments on the LINEMOD Dataset

The LINEMOD dataset contains 15 objects. We train and test our method on 13 of them as other methods in the literature. We follow the procedure in [10] to split the dataset into the training and test sets, with around 200 images for each object in the training set and 1,000 images in the test set.

Training strategy: For every image, we generate 10 random poses near the ground truth pose, resulting in 2,000 training samples for each object in the training set. Furthermore, we generate 10,000 synthetic images for each object where the pose distribution is similar to the real training set. For each synthetic image, we generate 1 random pose near its ground truth pose (see Appendix B.1). Thus, we have a total of 12,000 training samples for each object in training. The background of a synthetic image is replaced with a randomly chosen indoor image from PASCAL VOC [25]. We train the networks for 8 epochs with initial learning rate 0.0001. The learning rate is divided by 10 after the 4th and 6th epoch, respectively.

Ablation study on iterative training and testing: Table 1 shows the experimental results that use different numbers of iterations during training and testing (see Appendix D for the evaluation metrics for 6D pose estimation). The networks in the experiments with $train\_iter = 1$ and $train\_iter = 2$ are trained with 32 and 16 epochs respectively, in order to keep the total number of updates the same as $train\_iter = 4$. The time to decrease the learning rate is also changed correspondingly.

The table shows that without iterative training ($train\_iter = 1$), performing multiple iterations during testing does not improve the performance, potentially even making the results worse ($test\_iter = 4$). We believe that the reason is due

Table 1: Ablation study of the number of iterations during training and testing. The table shows the results if we change the number of iteration during training and during testing.

train iter	init	1			2			4		
test iter		1	2	4	1	2	4	1	2	4
5cm 5°	19.4	57.4	58.8	54.6	76.3	86.2	86.7	70.2	83.7	85.2
6D Pose	62.7	77.9	79.0	76.1	83.1	88.7	89.1	80.9	87.6	88.6
Proj. 2D	70.2	92.4	92.6	89.7	96.1	97.8	97.6	94.6	97.4	97.5

to the fact that the network is not trained with enough rendered poses close to their ground truth poses.

The table also shows that one more iteration during training and testing already improves the results by a large margin. In the table, the network trained with 2 iterations and tested with 2 iterations is slightly better than the one trained with 4 iterations and tested with 4 iterations. This may because the LINEMOD dataset is not sufficiently difficult to generate further improvements by using 3 or 4 iterations. Since it is not straightforward to determine how many iterations each dataset requires, we use 4 iterations during training and testing in all other experiments.

Ablation study on the zoom in strategy, network structures, transformation representations and loss functions: Table 2 summarizes the ablation studies on various aspects of DeepIM. The "zoom" column indicates whether the network uses full images as its input or zoomed in bounding boxes up-sampled to the original image size. Comparing rows 5 and 7 shows that the higher resolution achieved via zooming in provides very significant improvements.

"Regressor": We train the DeepIM network jointly over all objects in the dataset, generating a pose transformation independent of the specific input object (labeled "shared" in "regressor" column). Alternatively, we could train a different 6D pose regressor for each individual object by using a separate fully connected layer for each object after the final FC256 layer shown in Fig. 3. This setting is labeled as "sep." in Table. 2 . Comparing rows 3 and 7 demonstrates that the shared regression provides slightly better results by sharing training data across objects.

"Network": Similarly, instead of training a single network over all objects, we could train separate networks, one for each object as in [5]. Comparing row 1 to 7 shows that a single, shared network provides better results than individual ones, which indicates that training on multiple objects can help the network learn a more general representation for matching.

"Coordinate": This column investigates the impact of our choice of coordinate frame to reason about object transformations, as described in Sec. 3.3. The row labeled "naive" provides results when choosing the camera frame of reference as the representation for the object pose, rows labeled "model" move the center of rotation to the object model pose and choose the object model

Table 2: Ablation study on different design choices of the DeepIM network on the LINEMOD dataset.

Row	methods					5cm 5°	6D Pose	Proj. 2D
	zoom	regressor	network	coordinate	loss			
1	✓	-	sep.	camera	PM	83.3	87.6	96.2
2	✓	sep.	shared	model	PM	79.2	87.5	95.4
3	✓	sep.	shared	camera	PM	86.6	89.5	96.7
4		shared	shared	naive	PM	16.6	44.3	62.5
5		shared	shared	camera	PM	38.3	65.2	80.8
6	✓	shared	shared	camera	Dist	86.5	79.2	96.2
7	✓	shared	shared	camera	PM	85.2	88.6	97.5

Table 3: Ablation study on two different methods for generating initial poses on the LINEMOD dataset.

method	PoseCNN	PoseCNN +OURS	Faster R-CNN	Faster R-CNN +OURS
5cm 5°	19.4	85.2	11.9	83.4
6D Pose	62.7	88.6	33.1	86.9
Proj. 2D	70.2	97.5	20.9	95.7

coordinate frame to reason about rotations, and the "camera" rows provide our approach of moving the center into the object pose while keeping the camera coordinate frame for rotations. Comparing rows 2 and 3 shows that reasoning in the camera rotation frame provides slight improvements on this dataset. Furthermore, it should be noted that only our "camera" approach is able to operate on unseen objects. Comparing rows 4 and 5 shows the large improvements our representation achieves over the naive approach of reasoning fully in the camera frame of reference.

"Loss": The traditional loss for pose estimation is specified by the distance ("Dist") between the estimated and ground truth 6D pose coordinates, i.e., angular distance for rotation and euclidean distance for translation. Comparing rows 6 and 7 indicates that our point matching loss ("PM") provides significantly better results especially on the 6D pose metric, which is the most important measure for reasoning in 3D space.

Application to different initial pose estimation networks: Table 3 provides results when we initialize DeepIM with two different pose estimation networks. As we can see, our network achieves very similar pose estimation accuracy even when initialized with the estimates from an extension of Faster R-CNN [26], which are not as accurate as those provided by PoseCNN [8] (see Appendix C for details on Faster R-CNN based initial poses). Notice that the same network is used for refinement here with the two types of initial poses.

Table 4: Comparison with state-of-the-art methods on the LINEMOD dataset

methods	[10]	BB8 w ref.[5]	SSD-6D w ref.[14]	Tekin et al.[6]	PoseCNN[8]	PoseCNN[8] +OURS
5cm 5°	40.6	69.0	-	-	19.4	**85.2**
6D Pose	50.2	62.7	79	55.95	62.7	**88.6**
Proj. 2D	73.7	89.3	-	90.37	70.2	**97.5**

Comparison with the state-of-the-art 6D pose estimation methods: Table 4 shows the comparison with the best color-only techniques on the LINEMOD dataset. As we can see, DeepIM achieves very significant improvements over all prior methods, even those that also deploy refinement steps (BB8 [5] and SSD-6D [14]).

4.3 Experiments on the Occlusion LINEMOD Dataset

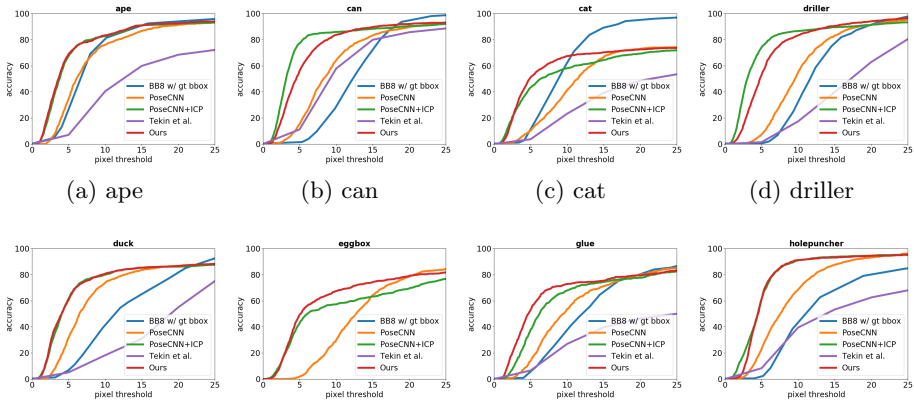
The Occlusion LINEMOD dataset, proposed in [10], shares the same images used in LINEMOD [7], but annotated 8 objects in one video that are heavily occluded.

Training: For every real image, we generate 10 random poses exactly the same way as in Sec. 4.2. Considering the fact that most of the training data lacks occlusions, we generated about 20,000 synthetic images with multiple objects in each image. By doing so, every object have around 12,000 images which are partially occluded, and a total of 22,000 images for each object in training. We perform the same background replacement and training procedure as in the LINEMOD dataset.

Comparison with the state-of-the-art methods: The comparison between our method and other RGB-only methods is shown in Fig. 4. We only show the plots with accuracies on the 2D Projection metric because these are the only results reported in [5] and [6] (results for eggbox and glue use a symmetric version of this accuracy). It can be seen that our method greatly improves the pose accuracy generated by PoseCNN and surpasses all other RGB-only methods by a large margin. It should be noted that BB8 [5] achieves the reported results only when using ground truth bounding boxes during testing. Our method is even competitive with the results that use depth information and ICP to refine the estimates of PoseCNN. Fig. 5 shows some pose refinement results from our method on the Occlusion LINEMOD dataset.

4.4 Application to Unseen Objects

As stated in Sec 3.3, we designed the untangled pose representation such that it is independent of the coordinate frame of a specific 3D object model. Therefore, the pose transformations correspond to operations in the image space. They are



(a) ape (b) can (c) cat (d) driller

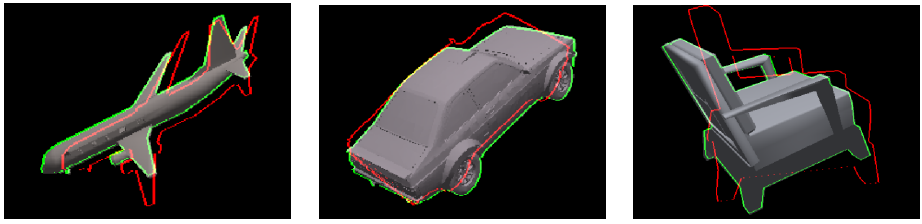
(e) duck (f) eggbox (g) glue (h) holepuncher

Fig. 4: Comparison with state-of-the-art RGB or RGB-D based methods on the Occlusion LINEMOD dataset [10]. Accuracies are measured via the Projection 2D metric, which measures the pixel difference between ground truth and estimated object poses.

independent of the absolute size of an object. This opens the question: whether DeepIM can refine the poses of objects that are not included in the training set. In this experiment, we use the 3D models of airplanes, cars and chairs from the ModelNet dataset [27]. For each of these categories, we train a network on no more than 200 3D models and test its performance on 70 unseen 3D models from the same category. For training, we generate 50 images for each model and train the network for 4 epochs. We found that our network can perform accurate refinement on these unseen models. See Fig. 6 for example results, more details and examples are provided in Appendix G. Appendix H shows that our network is also able to match unseen object categories in the training set.

(a) driller (b) can (c) ape

Fig. 5: Examples of refined poses on the Occlusion LILNEMOD dataset using the results from PoseCNN [8] as initial poses. The red and green lines represent the edges of the initial estimates and our refined poses, respectively.



(a) airplane (b) car (c) chair

Fig. 6: Results on pose refinement of 3D models from the ModelNet dataset. These instances were not seen in training. The red and green lines represent the edges of the initial estimates and our refined poses.

5 Conclusion

In this work, we introduce a novel DeepIM framework for iterative pose matching using color images only. Given an initial 6D pose estimation of an object, we have designed a new deep neural network to directly output a relative pose transformation that can be applied to the initial pose to improve 6D pose estimation. The network automatically learns to match object poses during training. A new untangled pose representation is also introduced that is independent of the coordinate frame of the 3D object model. In this way, the network can even match poses of unseen object instances as shown in our experiments. Our method significantly outperforms the state-of-the-art 6D pose estimation methods using color images only. The performance of our method is already close to methods that use depth images for pose refinement such as using the iterative closest point algorithm. These experimental results demonstrate the ability of our method to use color images for 6D pose matching.

This work opens up various directions for future research. For instance, we expect that a stereo version of DeepIM could further improve pose accuracy. Furthermore, DeepIM indicates that it is possible to produce accurate 6D pose estimates using color images only, enabling the use of cameras that capture high resolution images at high frame rates with a large field of view, providing estimates useful for applications such as robot manipulation.

Acknowledgements

This work was funded in part by a Siemens grant. We would also like to thank NVIDIA for generously providing DGX used for this research via the NVIDIA Robotics Lab and the UW NVIDIA AI Lab (NVAIL).

References

1. Lowe, D.G.: Object recognition from local scale-invariant features. In: IEEE International Conference on Computer Vision (ICCV). Volume 2. (1999) 1150–1157

2. Rothganger, F., Lazebnik, S., Schmid, C., Ponce, J.: 3D object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints. International Journal of Computer Vision (IJCV) **66**(3) (2006) 231–259
3. Collet, A., Martinez, M., Srinivasa, S.S.: The MOPED framework: Object recognition and pose estimation for manipulation. International Journal of Robotics Research (IJRR) **30**(10) (2011) 1284–1306
4. Brachmann, E., Krull, A., Michel, F., Gumhold, S., Shotton, J., Rother, C.: Learning 6D object pose estimation using 3D object coordinates. In: European Conference on Computer Vision (ECCV). (2014)
5. Rad, M., Lepetit, V.: BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth. In: IEEE International Conference on Computer Vision (ICCV). (2017)
6. Tekin, B., Sinha, S.N., Fua, P.: Real-time seamless single shot 6D object pose prediction. arXiv preprint arXiv:1711.08848 (2017)
7. Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., , Navab, N.: Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. In: Asian Conference on Computer Vision (ACCV). (2012)
8. Xiang, Y., Schmidt, T., Narayanan, V., Fox, D.: PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. arXiv preprint arXiv:1711.00199 (2017)
9. Tjaden, H., Schwanecke, U., Schömer, E.: Real-time monocular pose estimation of 3D objects using temporally consistent local color histograms. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2017) 124–132
10. Brachmann, E., Michel, F., Krull, A., Ying Yang, M., Gumhold, S., Rother, C.: Uncertainty-driven 6D pose estimation of objects and scenes from a single RGB image. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2016) 3364–3372
11. Michel, F., Kirillov, A., Brachmann, E., Krull, A., Gumhold, S., Savchynskyy, B., Rother, C.: Global hypothesis generation for 6D object pose estimation. IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
12. Zeng, A., Yu, K.T., Song, S., Suo, D., Walker, E., Rodriguez, A., Xiao, J.: Multi-view self-supervised deep learning for 6D pose estimation in the amazon picking challenge. In: IEEE International Conference on Robotics and Automation (ICRA). (2017) 1386–1383
13. Krull, A., Brachmann, E., Michel, F., Ying Yang, M., Gumhold, S., Rother, C.: Learning analysis-by-synthesis for 6D pose estimation in RGB-D images. In: IEEE International Conference on Computer Vision (ICCV). (2015) 954–962
14. Kehl, W., Manhardt, F., Tombari, F., Ilic, S., Navab, N.: SSD-6D: Making rgb-based 3D detection and 6D pose estimation great again. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2017) 1521–1529
15. Girshick, R.: Fast R-CNN. In: IEEE International Conference on Computer Vision (ICCV). (2015) 1440–1448
16. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2015) 3431–3440
17. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: European Conference on Computer Vision (ECCV). (2016) 21–37

18. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: IEEE conference on Computer Vision and Pattern Recognition (CVPR). (2016) 779–788
19. Carreira, J., Agrawal, P., Fragkiadaki, K., Malik, J.: Human pose estimation with iterative error feedback. In: IEEE conference on Computer Vision and Pattern Recognition (CVPR). (2016)
20. Oberweger, M., Wohlhart, P., Lepetit, V.: Training a feedback loop for hand pose estimation. In: IEEE International Conference on Computer Vision (ICCV). (2015)
21. Mousavian, A., Anguelov, D., Flynn, J., Košecká, J.: 3D bounding box estimation using deep learning and geometry. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2017) 5632–5640
22. Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., van der Smagt, P., Cremers, D., Brox, T.: Flownet: Learning optical flow with convolutional networks. In: IEEE International Conference on Computer Vision (ICCV). (2015) 2758–2766
23. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
24. Kendall, A., Cipolla, R.: Geometric loss functions for camera pose regression with deep learning. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2017)
25. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. International Journal of Computer Vision (ICCV) **88**(2) (2010) 303–338
26. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems (NIPS). (2015)
27. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3D shapenets: A deep representation for volumetric shapes. In: IEEE conference on Computer Vision and Pattern Recognition (CVPR). (2015) 1912–1920
28. Shotton, J., Glocker, B., Zach, C., Izadi, S., Criminisi, A., Fitzgibbon, A.: Scene coordinate regression forests for camera relocalization in RGB-D images. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2013) 2930–2937

A Details about Zooming In

Given the rendered mask $\mathbf{m}_{\text{rend}}$ and the observed mask $\mathbf{m}_{\text{obs}}$, the cropping patch is computed as Eq. 6

$$\begin{aligned} y_{\text{dist}} &= \max(|u_{\text{obs}} - y_c|, |u_{\text{rend}} - y_c|, \\ &\qquad |d_{\text{obs}} - y_c|, |d_{\text{rend}} - y_c|) \\ x_{\text{dist}} &= \max(|l_{\text{obs}} - x_c|, |l_{\text{rend}} - x_c|, \\ &\qquad |r_{\text{obs}} - x_c|, |r_{\text{rend}} - x_c|) \\ \text{width} &= \max(x_{\text{dist}}, y_{\text{dist}} \cdot r) \cdot 2\lambda \\ \text{height} &= \max(x_{\text{dist}}/r, y_{\text{dist}}) \cdot 2\lambda \end{aligned} \tag{6}$$

where u_*, d_*, l_*, r_* denotes the upper, lower, left, and right boundary of the foreground mask of observed or rendered images, x_c, y_c represent the 2D projection of the center of the object in $\mathbf{x}_{\text{rend}}$, r represent the aspect ratio of the origin image (width/height), λ denotes the expand ratio, which is fixed to 1.4 in the experiment. Then this patch is bilinear sampled to the size of the original image, which is 480×640 in this paper. By doing so, not only the object is zoomed in without being distorted, but also the network is provided with the information about where the center of the object lies.

B Details about Training Data

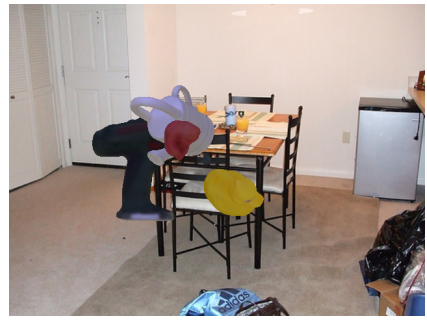
B.1 Estimated Pose during Training

The rendered image $\mathbf{x}_{\text{rend}}$ and mask $\mathbf{m}_{\text{rend}}$ are randomly generated during training without using prior knowledge of the initial poses in the test set. Specifically, given a ground truth pose $\hat{\mathbf{p}}$, we add noise to $\hat{\mathbf{p}}$ to generate the rendered poses. For rotation, we independently add a Gaussian noise $\mathcal{N}(0, 15^2)$ to each of the three Euler angles of the rotation. If the angular distance between the new pose and the ground truth pose is more than 45°, we discard the new pose and generate another one in order to make sure the initial pose for refinement is within 45° of the ground truth pose during training. For translation, considering the fact that RGB-based pose estimation methods usually have larger variance on depth estimation, the following Gaussian noise is added to the three components of the translation: $\Delta x \sim \mathcal{N}(0, 0.01^2), \Delta y \sim \mathcal{N}(0, 0.01^2), \Delta z \sim \mathcal{N}(0, 0.05^2)$, where the variances are 1 cm, 1 cm and 5 cm, respectively.

B.2 Synthetic Training Data

Real training images provided in existing datasets may be highly correlated or lack images in certain situations such as occlusions between objects. Therefore, generating synthetic training data is essential to enable the network to deal with different scenarios in testing. In generating synthetic training data for the LINEMOD dataset, considering the fact that the elevation variation is limited

(a) Synthetic Data for LINEMOD



(b) Synthetic Data for OCCLUSION

Fig. 7: Synthetic Data for the LINEMOD or Occlusion dataset. 7a shows the synthetic training data used when training on the LINEMOD dataset, only one object is presented in the image so there is no occlusion. 7b shows the synthetic training data used when training on the Occlusion dataset, multiple objects are presented in one image so one object may be occluded by other objects.

in this dataset, we calculate the elevation range of the objects in the provided training data. Then we rotate the object model with a randomly generated quaternion and repeat it until the elevation is within this range. The translation is randomly generated using the mean and the standard deviation computed from the training set. During training, the background of the synthetic image is replaced by a randomly chosen indoor image from the PASCAL VOC dataset as shown in Fig. 7.

For the Occlusion dataset, 3-8 objects are rendered into one image in order to introduce occlusions among objects. As in the LINEMOD dataset, the quaternion of each object is also randomly generated to ensure that the elevation range is within that of training data in the Occlusion dataset. The translations of the objects in the same image are drawn according to the distributions of the objects in the YCB-Video dataset [8] by adding a small Gaussian noise.

The real training images may also lack variations in light conditions exhibited in the real world or in the testing set. Therefore, we add a random light condition to each synthetic image in both the LINEMOD dataset and the Occlusion dataset.

C Pose Initialization

Our framework takes an input image and an initial pose estimation of an object in the image as inputs, and then refines the initial pose iteratively. In our experiments, we have tested two pose initialization methods.

The first one is PoseCNN [8], a convolutional neural network designed for 6D object pose estimation. PoseCNN performs three tasks for 6D pose estimation, i.e., semantic labeling to classify image pixels into object classes, localizing the center of the object on the image to estimate the 3D translation of the object, and

3D rotation regression. In our experiments, we use the 6D poses from PoseCNN as initial poses for pose refinement.

To demonstrate the robustness of our framework on pose initialization, we have implemented a simple 6D pose estimation method for pose initialization, where we extend the Faster R-CNN framework designed for 2D object detection [26] to 6D pose estimation. Specifically, we use the bounding box of the object from Faster R-CNN to estimate the 3D translation of the object. The center of the bounding box is treated as the center of the object. The distance of the object is estimated by maximizing the overlap of the projection of the 3D object model with the bounding box. To estimate the 3D rotation of the object, we add a rotation regression branch to Faster R-CNN as in PoseCNN. In this way, we can obtain a 6D pose estimation for each detected object from Faster R-CNN.

In our experiments on the LINEMOD dataset, we have shown that, although the initial poses from Faster R-CNN are worse than the poses from PoseCNN, our framework is still able to refine these poses using the *same* weights. The performance gap between using the two different pose initialization methods is quite small, which demonstrates the ability of our framework in using different methods for pose initialization.

D Evaluation Metrics

$n°$, n cm Proposed in [28]. The 5°, 5cm metric considers an estimated pose to be correct if its rotation error is within 5° and the translation error is below 5cm. We also provided the results with 2°, 2cm and 10°, 10cm in Table 5 to give a comprehensive view about the performance.

For symmetric objects such as eggbox and glue in the LINEMOD dataset, we compute the rotation error and the translation error against all possible ground truth poses with respect to symmetry and accept the result when it matches one of these ground truth poses.

6D Pose Hintertoisser et al. [7] use the average distance (ADD) metric to compute the averaged distance between points transformed using the estimated pose and the ground truth pose as in Eq. 7:

$$\mathbf{ADD} = \frac{1}{m} \sum_{\mathbf{x} \in \mathcal{M}} \|(\mathbf{R}\mathbf{x} + \mathbf{t}) - (\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{t}})\|, \tag{7}$$

where m is the number of points on the 3D object model, $\mathcal{M}$ is the set of all 3D points of this model, $\mathbf{p} = [\mathbf{R}|\mathbf{t}]$ is the ground truth pose and $\hat{\mathbf{p}} = [\hat{\mathbf{R}}|\hat{\mathbf{t}}]$ is the estimated pose. $\mathbf{R}\mathbf{x} + \mathbf{t}$ indicates transforming the point with the given SE(3) transformation (pose) $\mathbf{p}$. Following [10], we compute the distance between all pairs of points from the model and regard the maximum distance as the diameter d of this model. Then a pose estimation is considered to be correct if the computed average distance is within 10% of the model diameter. In addition

to using $0.1d$ as the threshold, we also provided pose estimation accuracy using thresholds $0.02d$ and $0.05d$ in Table 5.

For symmetric objects, we use the closet point distance in computing the average distance for 6D pose evaluation as in [7]:

$$\textbf{ADD-S} = \frac{1}{m} \sum_{\mathbf{x}_1 \in \mathcal{M}} \min_{\mathbf{x}_2 \in \mathcal{M}} \|(\mathbf{R}\mathbf{x}_1 + \mathbf{t}) - (\hat{\mathbf{R}}\mathbf{x}_2 + \hat{\mathbf{t}})\|. \tag{8}$$

2D Projection focuses on the matching of pose estimation on 2D images. This metric is considered to be important for applications such as augmented reality. We compute the error using Eq. 9 and accept a pose estimation when the 2D projection error is smaller than a predefined threshold:

$$\textbf{Proj. 2D} = \frac{1}{m} \sum_{\mathbf{x} \in \mathcal{M}} \|\mathbf{K}(\mathbf{R}\mathbf{x} + \mathbf{t}) - \mathbf{K}(\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{t}})\|, \tag{9}$$

where $\mathbf{K}$ denotes the intrinsic parameter matrix of the camera and $\mathbf{K}(\mathbf{R}\mathbf{x} + \mathbf{t})$ indicates transforming a 3D point according to the SE(3) transformation and then projecting the transformed 3D point onto the image. In addition to using 5 pixels as the threshold, we also show our results with the thresholds 2 pixels and 10 pixels.

For symmetric objects such as eggbox and glue in the LINEMOD dataset, we compute the 2D projection error against all possible ground truth poses and accept the result when it matches one of these ground truth poses.

E Detailed Results on the LINEMOD Dataset

Table 5 shows our detailed results on all the 13 objects in the LINEMOD dataset. The network is trained and tested with 4 iterations. Initial poses are estimated by PoseCNN [8].

F Detailed Results on the Occlusion LINEMOD Dataset

Table 6 shows our results on the Occlusion LINEMOD dataset. We can see that DeepIM can significantly improve the initial poses from PoseCNN. Notice that the diameter here is computed using the extents of the 3D model following the setting of [8] and other RGB-D based methods. Some qualitative results are shown in Figure 8.

G Test on Unseen Objects

Notice that the transformation predicted from the network does not need to have prior knowledge about the model itself. In this experiment, we explore the ability of the network in refining poses of objects that has never been seen

Table 5: Results of using more detailed thresholds on the LINEMOD dataset

metric	(n°, n cm)			6D Pose			Projection 2D		
threshold	(2, 2)	(5, 5)	(10,10)	0.02d	0.05d	0.10d	2 px.	5 px.	10 px.
ape	37.7	90.4	98.0	14.3	48.6	77.0	92.2	98.4	99.6
benchvise	37.6	88.7	98.2	37.5	80.5	97.5	67.7	97.0	99.6
camera	56.1	95.8	99.2	30.9	74.0	93.5	86.3	98.9	99.7
can	58.0	92.8	99.0	41.4	84.3	96.5	98.6	99.7	99.8
cat	33.5	87.6	97.8	17.6	50.4	82.1	88.4	98.7	100.0
driller	49.4	92.9	99.1	35.7	79.2	95.0	64.2	96.1	99.4
duck	30.8	85.2	98.5	10.5	48.3	77.7	88.1	98.5	99.8
eggbox	32.1	63.9	94.5	34.7	77.8	97.1	53.4	96.2	99.6
glue	32.8	83.0	98.0	57.3	95.4	99.4	81.5	98.9	99.7
holepuncher	8.7	54.5	93.8	5.3	27.3	52.8	59.1	96.3	99.5
iron	47.5	92.7	99.3	47.9	86.3	98.3	67.4	97.2	99.9
lamp	47.5	90.9	98.4	45.3	86.8	97.5	60.0	94.2	99.0
phone	34.8	89.6	98.6	22.7	60.5	87.7	75.9	97.7	99.8
MEAN	39.0	85.2	979	30.9	69.2	88.6	75.6	97.5	99.7

Table 6: Results on the Occlusion LINEMOD dataset. The network is trained and tested with 4 iteration.

metric	(5°, 5cm)		6D Pose 0.1d		Projection 2D 5 px.	
method	Initial	Refined	Initial	Refined	Initial	Refined
ape	2.1	**51.8**	9.9	**59.2**	34.6	**69.0**
can	4.1	**35.8**	45.4	**63.5**	15.1	**56.1**
cat	0.3	**12.8**	0.8	**26.2**	10.4	**50.9**
driller	2.5	**45.2**	41.6	**55.6**	7.4	**52.9**
duck	1.8	**22.5**	19.5	**52.4**	31.8	**60.5**
eggbox	0.0	**17.8**	24.5	**63.0**	1.9	**49.2**
glue	0.9	**42.7**	46.2	**71.7**	13.8	**52.9**
holepuncher	1.7	**18.8**	27.0	**52.5**	23.1	**61.2**
MEAN	1.7	**30.9**	26.9	**55.5**	17.2	**56.6**

Table 7: Results on unseen objects. These models are not included in the training set.

category	airplane		car		chair	
method	Initial	Refined	Initial	Refined	Initial	Refined
5cm 5°	0.8	**68.9**	1.0	**81.5**	1.0	**87.6**
6D Pose 0.1d	25.7	**94.7**	10.8	**90.7**	14.6	**97.4**
Proj. 2D 5 px.	0.4	**87.3**	0.2	**83.9**	1.5	**88.6**

Fig. 8: Some pose refinement results on the Occlusion dataset. The red and green lines represent the edges of 3D model projected from the initial poses and our refined poses respectively.

in training. ModelNet [27] contains a large number of 3D models in different object categories. Here, we tested our network on three of them: *airplane, car and chair*, where we train the network on around 100 object models and test the trained network on another unseen 100 object models. Similar to the way that we generate synthetic data as described in Appendix B.2, we generate 50 poses for each model as the target poses. We use uniform gray texture for each model and add a light source which has a fixed relative position to the object to reflect the norms of the object. The initial pose used in training and testing is generated in the same way as we did in previous experiments as described in Appendix B.1. The results are show in Table 7.

H Test on Unseen Categories

We also tested our framework on refining the poses of unseen object categories, where the training categories and the test categories are completely different. We train the network on 8 categories from ModelNet [27]: *airplane, bed, bench, car, chair, piano, sink, toilet* with 30 models in each category and 50 image pairs for

Table 8: Results on unseen categories. Those categories has never been seen by the network during training.

metric	(5°, 5cm)		6D Pose 0.1d		Projection 2D 5 px.	
method	Initial	Refined	Initial	Refined	Initial	Refined
bathtub	0.9	**71.6**	11.9	**88.6**	0.2	**73.4**
bookshelf	1.2	**39.2**	9.2	**76.4**	0.1	**51.3**
guitar	1.2	**50.3**	9.6	**69.6**	0.2	**77.1**
range hood	1.0	**69.8**	11.2	**89.6**	0.0	**70.6**
sofa	1.2	**82.7**	9.0	**89.5**	0.1	**94.2**
wardrobe	1.4	**62.7**	12.5	**79.4**	0.2	**70.0**
tv stand	1.2	**73.6**	8.8	**92.1**	0.2	**76.6**

each model. The network was trained with 4 iterations and 4 epochs. Then we tested the network on 7 other categories: *bathtub, bookshelf, guitar, range hood, sofa, wardrobe, and tv stand*. The results are shown in Table. 8. It shows that the network indeed has learned a general representation for pose refinement across different object categories.