FastPoseCNN: Real-Time Monocular Category-Level Pose and Size Estimation Framework

Eduardo Davalos
Vanderbilt University
Nashville, TN
eduardo.davalos.anaya@vanderbilt.edu

Mehran Aminian
St. Mary's University
San Antonio, TX
maminian@stmarytx.edu

Abstract

The primary focus of this paper is the development of a framework for pose and size estimation of unseen objects given a single RGB image - all in real-time. In 2019, the first category-level pose and size estimation framework was proposed alongside two novel datasets called CAMERA and REAL. However, current methodologies are restricted from practical use because of its long inference time (2-4 fps). Their approach's inference had significant delays because they used the computationally expensive MaskedRCNN framework and Umeyama algorithm. To optimize our method and yield real-time results, our framework uses the efficient ResNet-FPN framework alongside decoupling the translation, rotation, and size regression problem by using distinct decoders. Moreover, our methodology performs pose and size estimation in a global context - i.e., estimating the involved parameters of all captured objects in the image all at once. We perform extensive testing to fully compare the performance in terms of precision and speed to demonstrate the capability of our method.

Keywords 6D pose estimation · monocular · real-time · neural network

1 Introduction

We study rigid-body 6D pose and size estimation to detect and recognize object's spatial information which includes translation, rotation, and scale. With all this information, we can pinpoint the object in 3D space and understand its relationship to the surrounding environment. This technology provides a foundation for various practical applications, including mixed reality, robotics, and object tracking. Historically, methods in 6D pose and size estimation were focused on feature matching through the use of algorithmic feature extractors such as SIFT (Scale-Invariant Feature Transform) [1] and SURF (Speeded-Up Robust Features) [2].

Pose and size estimation, similar to other computer vision (CV) tasks, has undergone a complete transformation with the emergence of deep learning and convolutional neural networks (CNN). In recent years, most proposed state-of-the-art methods have used end-to-end neural network models whose input is an image, and their output is the final pose and size of targeted objects.

In this work, RGB images are used as input to make multiple dense pixel-wise predictions, including the centroid, quaternion, and size vector fields and depth regression. These dense outputs are converted into instance-wise attributes through an aggregation routine that allows the estimation of the final pose and size. The regress parameters are constructed from the collective information of an object's seen pixels through the use of pixel-wise data, resulting in robustness against occlusion and truncation. The code for this model which is open source can be found in the following GitHub repository: https://github.com/edavalosanaya/FastPoseCNN.

The contribution of this work to the field can be summarized as follows:

- Proposing a novel CNN model for category-level 6D pose and size estimation named FastPoseCNN. Our network generates dense pixel-wise predictions for each decoupled spatial component.
- Achieving real-time inference only requires RGB images as input, and it is robust to occlusion and truncation through its pixel-wise driven pose and size estimation.

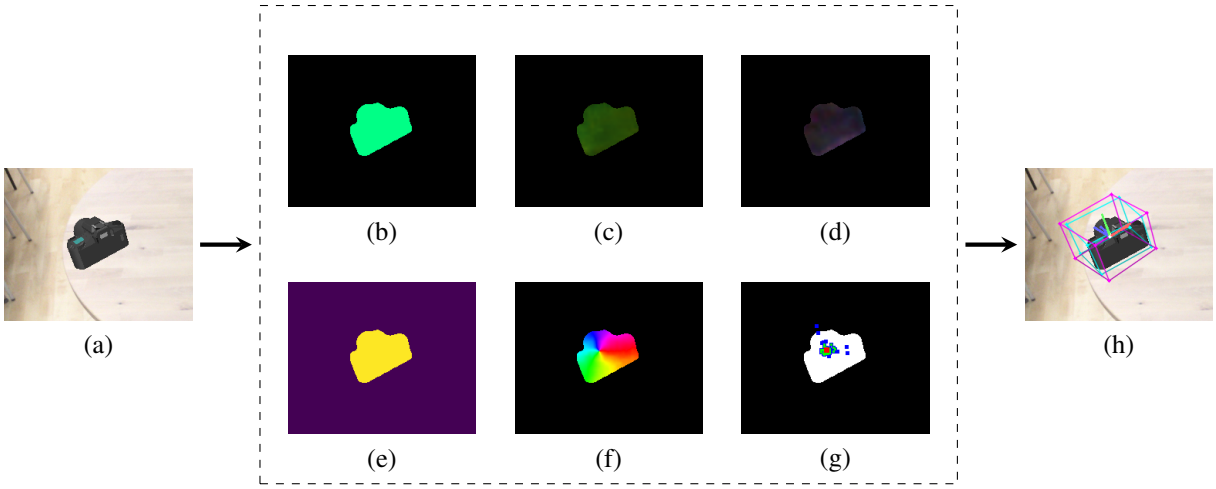


Figure 1: Model's Intermediate Data Representation. (a) Input RGB image, (b) segmentation, (c) quaternion, (d) size, (e) depth, (f) centroid vectors, (g) 3D centroid, (h) pose and size instance data.

- Introducing SymQuaternion-Loss, a new training loss function for quaternion regression that accounts for symmetric objects.

2 Literature Review

The field of 6D pose and size estimation has many different approaches and methodologies for estimating rigid body spatial information. We begin our overview of the literature by an initial categorization from previous works - primarily the problem and its solution constraints. These constraints are illustrated in the following subsections: 2.1 specifies input data types to a model, 2.2 describes object variation, and 2.3 elaborates the throughput requirement of solutions. Afterward, we will discuss the types of strategies and approaches used in the literature in Section 2.4 and how these methods compare to each other in terms of performance, efficiency, and usability. Finally, we conclude our literature review by addressing the gap mentioned in Section 2.5 using our method.

2.1 Input Data Type Differences

The first distinction among many proposed solutions is the type of input image or information they used. Standard data inputs include RGB images, RGBD images, depth images, point clouds, and other input representations. Different data representations provide unique challenges, complexities, and constraints.

RGB Image Input. Methods that used RGB images have gained high attention, mostly due to the commonality of RGB cameras in modern devices. Determining the 6D pose and size of an object from a single RGB image is inherently more challenging than using an RGBD image. It is because the missing depth information introduces perspective ambiguity. Even for humans, an object's unknown depth results in scaling ambiguities. Therefore, making pose and size estimation especially difficult.

RGBD Image Input. Historically, RGBD methods have been more commonly used in the 6D pose estimation field [3]. Methods that rely on RGBD input images utilize the 3D features to capture the pose of objects more accurately. These methods have achieved excellent performance by using neural networks and other machine learning approaches [4, 5, 6]. One of the downsides of this mode of input data is the expensive requirement of an RGB+depth camera.

Depth Image Input. Pure depth methods rely on the volumetric information provided in a depth image to estimate the pose [7, 8]. These methods use lighter machine learning models such as random forest models to make their inference extremely fast, e.g., 2 ms for [8]. These methods benefit from smaller yet efficient models as they require fewer training samples. The major drawback of these methods is their relatively lower performance, compared to RGB and RGBD methods, in more challenging datasets. The decreased performance is more visible when tested on the difficult LINEMOD-OCCLUDED and LINDEMOD-TRUNCATION datasets.

Point-Clouds Input. With the development of PointNet [9], Point clouds for CV methods have gained interest primarily because of their powerful integration of depth in a native 3D space. By lifting RGBD images into a point cloud representation, DL models benefit more from the present depth information by learning complex 3D features [10, 11, 12, 13]. Fusion methods combine the learned features from the point cloud and RGB image to better estimate an object's pose.

2.2 Category-Level vs. Instance-Level

(a) (b)

Figure 2: (a) Category-Level vs. (b) Instance-Level Dataset

The first modern datasets available for 6D pose estimation included LINEMOD [14], LINEMOD-OCCLUDED [3], YCB-V [15], and T-LESS [16]. All of these datasets share the property of being instance-level datasets. Instance-level implies that the dataset does not include any variants for objects of the same type. There is only one instance of each type in an object class. This is a major flaw in SOTA pose research since it does not accurately reflect the nature of objects found in reality. The discrepancy between real-life data and SOTA research datasets has led to the need for a 6D pose and size category-level dataset.

The landmark publication by [17] provided the first pair of publicly available category-level datasets, named CAMERA and REAL. CAMERA is an extensive synthetic dataset containing realistic backgrounds with rendered 3D models in workplace-related environments. REAL is a smaller real dataset with the same object categories and similar backgrounds. These datasets included a variety of distinct instances for each object category, as shown in Fig. 2. It added another level of complexity to pose estimation as now models had to account for these per-instance differences and how that might affect the effectiveness of methods.

Size Estimation Requirement

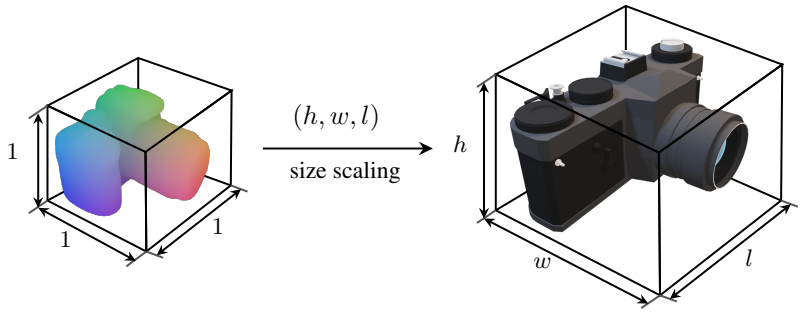


Figure 3: Size Regression

Additionally, [17] was the first paper to address category-level 6D pose and size estimation problem - specifically the addition of the size requirement. This is because instance-level pose estimation automatically provides the scale of an object since the exact 3D model of the object is known. However, the category-level problem adds the size requirement to fully estimate the complete bounding box of an object to account for per-instance size variations.

2.3 Real-Time Inference

Another major concern present in pose and size estimation is the inference time of solutions. Models need to detect and classify objects and their corresponding pose efficiently to allow applications to be built on top of the model; therefore,

this time efficiency requirement is a commonly sought feature in the CV. This was a problem for [17] as their proposed method was only able to run within 2-4 fps on an Intel Xeon Gold 5122 CPU @ 3.60GHz desktop with an NVIDIA TITAN Xp. The slow inference makes their MaskRCNN-NOCS model not practical for time-sensitive applications.

2.4 Related Work

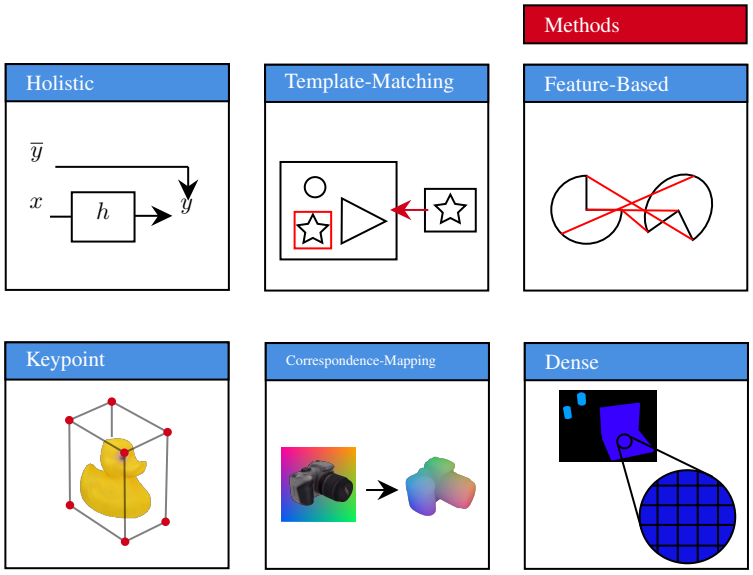


Figure 4: Types of Methods - Simply Illustrated

Holistic Methods. Holistic methods take the approach of directly regressing the pose, size, or other object's attributes. To simplify the nonlinearity of the rotation space, [18, 19, 20, 21] quantized the SO(3) space - making it into a more stable yet less accurate classification problem. It is common practice in the literature to take a mixed approach when estimating an object's 6D pose and size. [15] uses a CNN feature extractor to estimate the decoupled translation and rotation. First, the extracted features were used to estimate the translation via dense keypoint regression to identify the centroid (x, y) and dense pixel-wise regression for the direct depth (z). Second, they used the extracted features to directly regress the rotation by approximating an object's quaternion $\mathbf{q}$.

Template-matching Methods. Before the use of DL in pose estimation, template-matching was widely used to estimate the pose of rigid bodies [22, 23, 24, 25, 26, 27, 28, 29]. Template-matching methods use a sliding-window algorithm that calculates a similarity score between an image and multiple perspective-based templates. The major advantage in template-matching is its great ability to estimate the pose of texture-less objects with great performance. However, its heavy reliance on the similarity score reduces its performance when exposed to occlusion, truncation, and lighting variations.

Feature-based Methods. Another method used in the traditional field of 6D pose estimation, hand-crafted, and feature engineering were used for feature extraction and matching [30, 31, 32]. However, feature extraction and matching require texture-rich objects to accurately detect and recognize these objects. With the help of CNN's in pose estimation, using trainable end-to-end neural network models has become commonplace as these approaches learn more effective methods for extracting features in more challenging scenarios [15, 33, 34, 35, 10, 17, 11, 36]. After using features instead of templates, methods have become more robust to occlusion and truncation. Handling symmetries in objects have posed a greater challenge to feature-based methods in part because of symmetric-induced orientation ambiguities.

Keypoint-based Methods. Keypoint-based methods rely on regressing 2D keypoints of 3D objects instead of directly estimating the 3D translation and rotation. The use of keypoints as an intermediate representation of the pose and size stabilizes as well simplifies the learning problem. These methods use CNNs for feature extraction and segmentation to perform keypoint predictions through regions [34], heatmaps [37, 18], or pixel-wise predictions [33, 38]. Many approaches use hough voting and unit-vectors within keypoint-based methods to determine the keypoints corresponding

to objects' projected 3D bounding box edges or 3D centroid point [15, 38, 33]. While using Perspective-n-Point (PnP), these methods can obtain the pose and size from these 2D-3D correspondence keypoints.

Correspondence-Mapping Methods. Another data representation that connects 2D-3D spaces is the direct use of 2D-3D correspondence mapping of objects. Methods such as [17, 35, 39, 40, 3] utilize dense correspondence mapping to regress intermediate representations, such as 3D object coordinates, that aid in determining the rotation and translation of the objects. [17] proposed normalized object coordinate space to integrate size estimation in 3D coordinate regression.

Dense Methods. Dense methods utilize pixel-wise predictions that contribute via a reduction function to the overall object's pose. These methods have been used to regress keypoints, correspondence maps, and direct components of the pose, i.e rotation, translation, and size. Through reduction schemes, such as hough voting, RANSAC, and native averaging, dense predictions are converted into instance-wise predictions.

2.5 Addressing the Gap in the Literature

Our research focuses on addressing the disadvantages in the groundbreaking publication of [17] - i.e., slow inference and dependence on depth information. Their approach regressed dense correspondence mapping via a heavy MaskRCNN framework (210 ms) and later used the Umeyama algorithm (30 ms) for pose alignment. Their approach greatly benefited in performance by using correspondence mapping - yet this design decision slowed the model's speed. Our approach takes inspiration from RGB methods [38, 33, 15] by using the smaller ResNet-FPN framework and regressing both intermediate representations and direct attributes. Through these representations, our method allows the computation of the pose and size with greater computation efficiency. By making our method only use RGB images, we make our approach compatible with most modern cameras. This delay reduction and depth independence are at a small performance penalty while rendering our method useful for time-critical and hardware-limited applications.

3 Methodology

3.1 Framework Overview

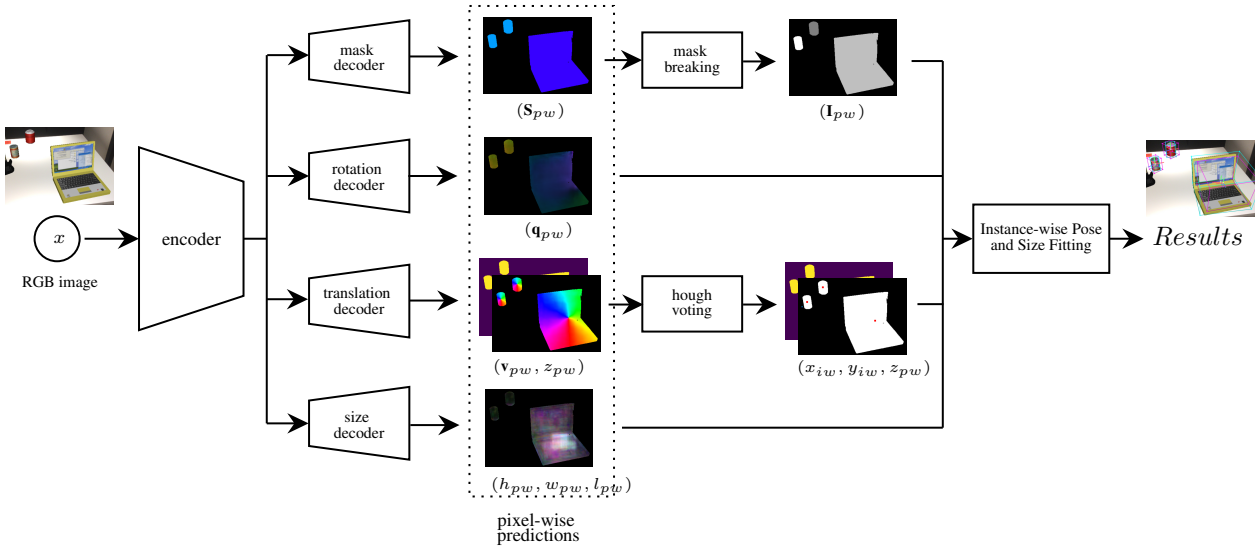


Figure 5: Model's Overall Architecture

In our approach to 6D pose and size estimation, we decoupled the regression for each spatial component of the objects. This lead to our architecture handling the classification, rotation, translation, and size estimation in separate decoders. They are demonstrated in our architecture outline in Fig. 5. The decoupling of these attributes ensures stability in training and better regression as each branch effectively adapts to the typical range of the designated data.

As shown in Fig. 5, our data pipeline is composed of CNN-generated pixel-wise predictions, mask breaking, hough voting, and aggregation. The input image's features are extracted by the encoder and then later used by the decoupled decoders. These decoders generate multiple dense pixel-wise predictions for segmentation, direct regression, and unit vector regression. The segmentation output is converted to instance masks via mask breaking. The unit vectors are used in hough voting to generate a centroid hypothesis for a detected object. After converting the intermediate data, aggregation takes place to match pixel-wise predictions to specific instances. In the following sections, each component of the data pipeline will be discussed in more detail.

3.2 Pixel-Wise Predictions

Our proposed method creates dense pixel-wise predictions for all parameters of the final pose. Our individual parameter decoders are inspired by [34, 35, 15, 33] - we noticed that small independent decoders improved stability and performance in training without a significant increase in inference time. The mask branch creates a segmentation mask. The rotation branch regresses dense pixel-wise quaternion predictions. We utilized quaternions here instead of rotation matrix due to the lower number of parameters to regress. This is to ensure that the problem space is smaller and less complex. The translation branch regressed both dense predictions for centroid unit vectors and the depth. The size branch generates dense (h, w, l) predictions.

By using dense pixel-wise predictions with the ResNet framework, our approach generates predictions for multiple objects in a single step - via a global context. Later in the process, the aggregation of these dense predictions is performed in an optimized batch manner - enabling the quick translation between pixel-wise to instance-wise pose and size information. Our method differs from other published works [17, 21] that perform 2D object detection and serially estimate an object's spatial attributes. Our method process multiple instances in parallel - thereby reducing the delay when multiple objects are captured in the image.

Class Masking and Compression. Our method outputs pixel-wise predictions for each class to ensure the data of different classes do not interfere and lower the model's performance. Afterward, we utilize the segmentation mask to compress the pixel-wise predictions and reduce their dimensionality to match categorical data. By doing this, our model can effectively estimate the pose and size of multiple classes without any speed penalty.

3.3 Segmentation Mask Breaking

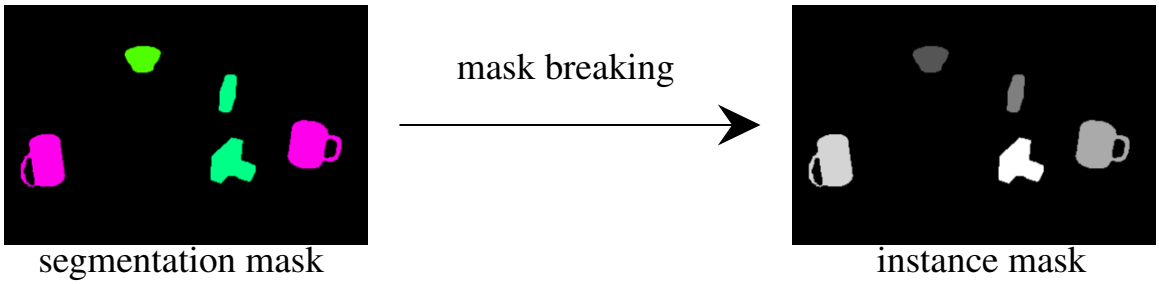


Figure 6: Mask Breaking

To match the pixel-wise information between all predictions, we convert the segmentation mask to a collection of instance masks. We utilize the GPU-accelerated implementation of scipy.ndimage.label provided in the CuPy library [41] to perform mask breaking. Once we convert the instance mask, it's matched with the corresponding dense pixel-wise to each object instance captured in an image by matching it with the instance mask.

Through our ResNet-FPN implementation, class segmentation and mask breaking improve the performance and shorten the training time of the model. It also allowed the optimization of the aggregation later down the pipeline. By breaking the segmentation mask into instance masks, the reduction function of the pixel-wise predictions can be performed in parallel for all instances captured in the instance masks.

3.4 Pixel-Wise Hough Voting

For our regression of objects' x and y translation attributes, we used the intermediate centroid unit vectors that point towards the projection of the 3D center of an object. To convert these unit vectors to proper translation parameters,

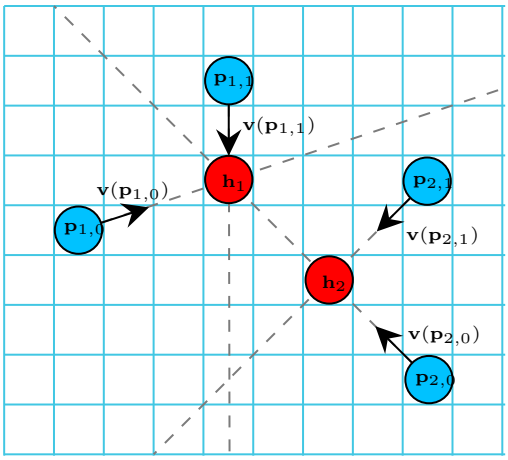


Figure 7: Hough Voting Scheme

we used the popular hough voting approach. For the hough voting scheme, we adapted the CUDA-accelerated implementation proposed by PVNet [38] for our problem as it provides a fast and accurate method to process multiple centroids in a batched fashion.

In our adapted hough voting algorithm, the pixel-wise centroid unit vectors are translated into a final $\mathbf{c} = (u, v)^T$ hypotheses within the image plane. The centroid unit vectors $\mathbf{v}(\mathbf{p})$ of a pixel $\mathbf{p}$ is defined by the relative location of the pixel $\mathbf{p}$ from the centroid $\mathbf{c}$, shown in Eq. 1. With this definition, the centroid unit vectors point towards the projected 3D centroid of objects.

$$\mathbf{v}(\mathbf{p}) = \frac{\mathbf{c} - \mathbf{p}}{||\mathbf{c} - \mathbf{p}||_2} \tag{1}$$

During the class masking and compression step, we apply bit-wise masking on pixel-wise unit vectors with the mask of the same classes. This step prepares the data to generate N number of hypotheses. As shown in Eq. 2, we construct a hypothesis, $\mathbf{h}$, for the projected centroid by obtaining the intersection between a random pair of unit vectors.

$$\mathbf{h}_i = \mathbf{v}(\mathbf{p}_{i,0}) \cap \mathbf{v}(\mathbf{p}_{i,1}) \tag{2}$$

Once we construct N number of hypotheses, we calculate the weights for each hypothesis by incorporating the rest of the object's unit vectors. These weights measure confidence in the matching hypothesis. The weights are calculated by counting the number of unit vectors that agree with the hypothesis - that is, that they point towards the hypothesis. In Eq. 3, the θ is a threshold (usually 0.99), O is the object's pertaining unit vector pixel-wise predictions.

$$w_i = \sum_{\mathbf{p} \in O} \boldsymbol{I} \left(\frac{(\mathbf{h}_i - \mathbf{p})^T}{||\mathbf{h}_i - \mathbf{p}||_2} \mathbf{v}(\mathbf{p}) \geq \theta \right) \tag{3}$$

The final centroid hypothesis is determined by calculating the weighted average of the N hypotheses. By using the hypothesis weights, the entirety of the object's unit vectors is included in the final hypothesis calculation. This contributes to faster learning with a smaller requirement of the number of hypotheses generated.

$$\mathbf{h}_{final} = \frac{\sum_{i=0}^{N} w_i \mathbf{h}_i}{\sum_{i=0}^{N} w_i} \tag{4}$$

The resulting $\mathbf{h}_{\text{final}}$ is later used, by combining it with the regressed depth z, to construct the translation vector $\mathbf{t}$ of an object. To completely reconstruct the pose and size parameters of instances captured by the model, we perform an aggregation step to convert the pixel-wise predictions to instance-wise, and this part will be elaborated in the next section.

3.5 Aggregation

With the creation of dense pixel-wise predictions for the pose and size variables, we need to compress dense predictions into instance-wise predictions to attach these parameters to captured objects. The overall aggregation routine is

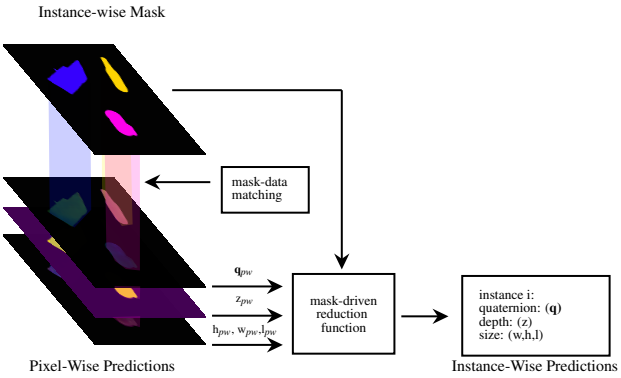


Figure 8: Aggregation via Masked Average

illustrated in Fig. 8. Our method utilizes the instance mask to individually extract the instance's information using a mask-drive reduction function. To ensure that our method remains real-time, we selected the fast and simple naive average of the masked dense predictions. The related reduction function is shown in Eq. 5 - where **a** is a placeholder for any pixel-wise predictions, O is the object's pixel-wise predictions, and the I is the binary instance mask.

$$\mathbf{a}_{aggregated} = \frac{\sum_{\mathbf{p} \in O} \mathbf{a}(\mathbf{p})}{\sum_{\mathbf{p} \in O} \boldsymbol{I}\left(I(\mathbf{p}) = 1\right))} \tag{5}$$

Here, we convert dense pixel-wise predictions into instance-wise predictions that allow us to create complete instance profiles that contain the translation, size, and rotation parameters. Except for the centroid unit vectors, these are not included in the aggregation routine - these pixel-wise predictions are handled by the hough voting step.

3.6 Ground Truth and Prediction Matching

During training, we focus on comparing the instance-wise predictions instead of the pixel-wise predictions. It shifts the focus of the optimization problem to the estimated final pose and size parameters rather than the intermediate dense pixel-wise predictions. Therefore, we matched the ground truth instances and the predicted instances using the 2D intersection over union (IoU) metric. By calculating all of the 2D IoU's between the ground truth and predicted instance masks, we matched the instances by assigning them to their corresponding highest 2D IoU score match. We noticed how direct optimization was more stable during our training process and prevented the model's estimation performance from degrading for smaller objects - due to their smaller pixel counts.

3.7 Loss Functions

Similar to our decoupling approach, we use separate loss functions for each branch of the model. Afterward, we sum the individual contribution of each loss to determine the total. Each loss is structured to account for the range and dimension of each parameter's problem space.

$$L_{total} = L_{mask} + L_{sym-quat} + L_{centroid} + L_{depth} + L_{scales} \tag{6}$$

Segmentation. The loss function used for segmentation is the summation between multi-class cross-entropy and focal [42] loss functions.

$$L_{mask} = L_{ce}(\overline{m}, m) + L_{focal}(\overline{m}, m) \tag{7}$$

Quaternion. For regressing the quaternion, we initially used QLoss, as shown in Eq. 8 as L_{quat}, referred in [43]. This loss function accounts for the internal symmetry of quaternions. However, it does not account for the symmetric object's axis of symmetry. We propose $L_{sym-quat}$ for making the quaternion loss function fit for symmetric objects.

$$L_{quat} = \log(\epsilon + 1 - |\overline{\mathbf{q}} \cdot \mathbf{q}|) \tag{8}$$

Symmetric Object Handling. A major issue when performing rotation estimation is symmetry. Many of the objects present in the CAMERA/REAL datasets have an axis of symmetry. We explored and determined that if we directly

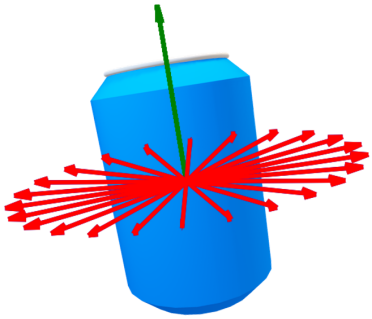
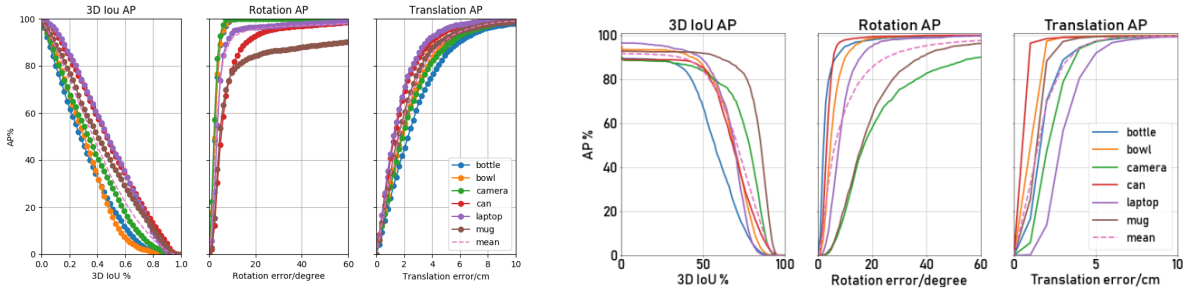


Figure 9: Quaternion Axis-of-Symmetry Rotation



(a) FastPoseCNN Performance Curves

(b) NOCSNet Performance Curves

Figure 10: 3D detection and 6D pose estimation results for CAMERA validation

regressed the rotation without considering the axis of symmetric of certain objects, the model yielded drastically lower performance for those objects.

As shown in Fig. 9, our strategy is to generate a set of equivalent ground truth quaternions rotated around the axis of symmetric. The goal is to capture all possible correct quaternion orientations and concentrate the quaternion optimization problem for symmetric objects to capture the axis of symmetry.

Therefore, we annotate the axis of symmetry for each type of symmetric object. We specify a set of rotation angles, $\theta = \{0°, 1°, ..., 359°\}$, which we use to form a new set of transformation quaternion, $\hat{\mathbf{q}}_i = \{\hat{\mathbf{q}}_{0°}, ..., \hat{\mathbf{q}}_{|\theta|}\}$. By applying the transformation quaternion to the ground truth quaternion $\overline{\mathbf{q}}_i = \hat{\mathbf{q}}_i \overline{\mathbf{q}} \hat{\mathbf{q}}_i^{-1}$, we construct ground truth quaternions, $\overline{\mathbf{q}}_i = \{\overline{\mathbf{q}}_{0°}, ..., \overline{\mathbf{q}}_{|\theta|}\}$ that capture all possible valid rotations. As shown in Eq. 9, we calculate the L_{quat} for each $\overline{\mathbf{q}}_i$ and use the lowest loss value.

$$L_{sym-quat} = \begin{cases} \min_{i=i,...,|\theta|} L_{quat}\left(\overline{\mathbf{q}}_i, \mathbf{q}\right) & \text{if symmetric} \\ L_{quat}\left(\overline{\mathbf{q}}, \mathbf{q}\right)) & \text{otherwise} \end{cases} \tag{9}$$

Centroid Unit Vectors. Early in the training phase, large quantities of outliers are generated by the untrained hough voting scheme. Through multiple trials, we determined that L1 outperformed smooth L1 and L2 in this task. Therefore, for regressing the projected 3D centroid, we use L1 loss on each coordinate space of the centroid.

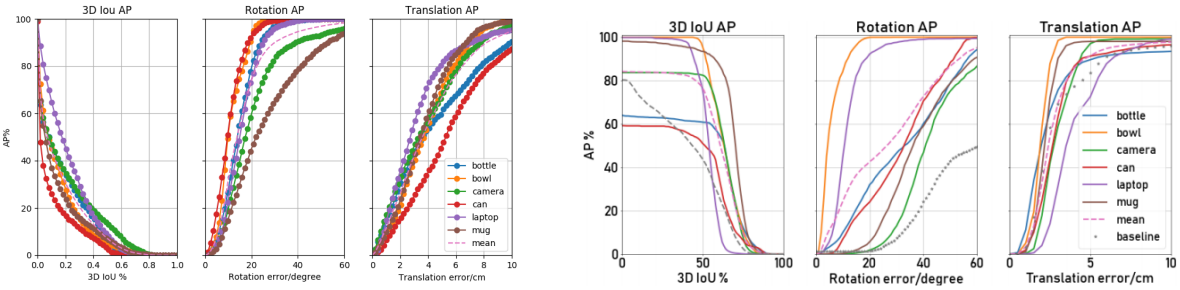
$$L_{centroid} = \ell_1(\overline{\mathbf{c}}|_x, \mathbf{c}|_x) + \ell_1(\overline{\mathbf{c}}|_y, \mathbf{c}|_y) \tag{10}$$

Depth. The ambiguity of estimating the depth makes this component of the model have the lowest performance. Additionally, minor errors in the depth estimation have significant negative effects on metrics such as 3D IoU thresholds. We follow the stabilizing technique proposed by [33] of estimating the $\log(z)$ instead of the z. This enhances the performance and stabilizes the training of the model.

$$L_{depth} = \ell_1(\log(\overline{z}), \log(z)) \tag{11}$$

Size Scales. For regressing the size of objects, we use L1 loss functions on each component of the size. This ensures the outliers in scales parameters were better handled compared to using smooth L1 or L2 loss functions.

$$L_{scales} = \ell_1(\overline{\mathbf{s}}|_h, \mathbf{s}|_h) + \ell_1(\overline{\mathbf{s}}|_w, \mathbf{s}|_w) + \ell_1(\overline{\mathbf{s}}|_l, \mathbf{s}|_l) \tag{12}$$



(a) FastPoseCNN Performance Curves

(b) NOCSNet Performance Curves

Figure 11: 3D detection and 6D pose estimation results for REAL test dataset

4 Experiments and Results

	CAMERA	
methods	NOCS	OURS
$3D_{25}$	90.13	66.69
$3D_{50}$	87.58	32.33
5°&5cm	38.14	67.16
10°&5cm	61.24	84.14
10°&10cm	62.01	91.02

(a) Validation on CAMERA25

Class	OURS			NOCS		
	$3D_{50}$	5°&5cm	10°&10cm	$3D_{50}$	5° & 5cm	10°&10cm
bottle	20.22	77.15	97.88	89.18	78.99	91.56
bowl	35.08	79.18	99.13	91.36	51.18	87.77
can	26.97	81.19	99.24	85.28	78.78	97.32
laptop	54.12	71.07	93.45	85.56	16.52	63.67
camera	39.53	51.88	76.90	83.59	2.01	17.47
mug	37.00	44.03	79.53	90.53	13.50	14.28

(b) Class-Wise performance on CAMERA25

Table 1: Additional breakdown comparison information in CAMERA.

4.1 Tools and Source Code

Our experiments and model were implemented using the PyTorch framework [44] and PyTorch-Lightning [45] open source libraries. We also used PyTorch Segmentation [46] pre-trained FPN-resnet18 model to jump start our training. Our model adapted PVNet's CUDA-accelerated hough voting scheme [38].

4.2 Implementation and Training

We initialize the ResNet18-FPN backbone with the weights pre-trained on Imagenet. In the first stage of training, we focus on training the mask branch of the model. We used a batch size of 2, an initial learning rate of 1×10^{-4}, and a RAdam optimizer with a weight decay of 3×10^{-4}. We disabled the hough voting, mask breaking, and aggregation steps that are later used in the second stage of training. After freezing all the layers except those found in the encoder and mask branch, we begin the speed-optimized training for 50 epochs. In the second stage of training, we trained the entire model for another 50 epochs. We used the same hyperparameters for this stage while enabling all intermediate steps of the model during training. The learning rate is reduced by a factor of 0.25 by a plateau scheduler in both stages.

4.3 Metrics

We adopt the metrics used in [17] to evaluate our results. These metrics include the 3D IoU and the mean average precision (mAP) where the error is less than m cm for translation and n° for rotation. By considering separate metrics for translation, rotation, and scale, we can more clearly present the performance of the model. For symmetric objects, we apply the same technique used in the symmetric loss function by rotating the ground truth quaternion and 3D bounding box by the axis-of-symmetric and selecting the highest performance value.

4.4 Comparison to SOTA Methods

As the following, we report and compare our category-level results to the reported values of NOCSNet [17] on both datasets. Due to our training structure, comparisons with NOCSNet for the CAMERA25 validation dataset use the reported data with the same training amount and type.

CAMERA25 Validation. We tested our model against the CAMERA25 validation set after only training with the 275K CAMERA training set. Our model achieves **32.33%** for 3D IoU at 50% and an $5°\&5$cm mAP of **66.69%**. The precision curves for these metrics are shown in Fig. 10. Note that these metrics are quite challenging because of the perspective ambiguities introduced by the unknown depth of objects.

REAL275 Testing. After training the model on the CAMERA275 training dataset, we train an additional 50 epochs specifically on the REAL training dataset. When tested on the REAL test dataset, our model performs **7.18%** for 3D IoU at 50% and an $5°\&5$cm mAP of **5.18%**. The precision curves for these metrics are shown in Fig. 11. The shifting of domains is especially tough when real data is not sufficient. Both NOCSNet and our proposed method have significantly decreased performance when testing on the REAL test dataset.

The approach proposed by [17] yields higher performance for the 3D IoU metric primarily because of its high-quality correspondence mapping method used to regress the object's scales and its use of depth images. However, correspondence mapping methods, like NOCSNet, do not handle well axis of symmetry. Our results verify this - as our direct regression rotation method achieves higher mAP. It is also important to consider that our method regresses the depth with good accuracy yet it negatively affects the 3D IoU metric. Our method achieves good performance while not requiring additional depth information.

In Table 1, we present our per-class performance in 1b and a further breakdown in performance in 1a. Through the class-wise information, it is clear that our method can capture the symmetric nature of objects better compared to [17] using our $L_{quat-sym}$ loss function.

4.5 Inference & Time-Breakdown

The primary purpose of this research is to provide a real-time monocular version of [17]. Our method can achieve an average delay of 43ms (23 fps) during inference - allowing our method to be considered real-time for pose estimation applications. A run-time breakdown is presented in Table 2. As far as we know, we are the first to propose a monocular category-level pose and size estimation framework with a real-time inference.

Component	Delay Time (ms)
Feature Extractor	18.570
Aggregation	4.808
Hough Voting	12.894
RT Calculation	3.769
Class Compression	2.660
Total	43.355 (23fps)

Table 2: Total Model Time-Breakdown

4.6 Limitations

In this section, we elaborate on the limitations of FastPoseCNN. First, the Hough Voting algorithm obtained from PVNet requires a CUDA-compatible GPU device to run it. Second, to use a different camera, additional training would be required. As referred by the pose estimation community, the camera intrinsics were baked into the model's parameters when we trained on the CAMERA and REAL datasets. This should not affect performance drastically, but it should be noticed. Third, the pose and size estimated by FastPoseCNN are excellent to a certain degree. FastPoseCNN should not be used for applications that depend on precision-critical objects' pose and size estimation. The research presented here is a proof-of-concept and would require further research and development to become a commercially reliable system.

5 Conclusion

In this thesis, we have introduced a real-time monocular category-level pose and size estimation framework that globally detects and estimates an object's pose and size parameters via dense pixel-wise predictions. FastPoseCNN is excellent at estimating the pose of symmetric objects while running in real-time. We showed how our specialized $L_{sym-quat}$ loss function improves the training of the model and outperforms NOCSNet in rotation estimation. Our experiment and analysis section demonstrates the performance and speed of FastPoseCNN compared to previous works. In future work, we plan on creating a more robust intermediate size and depth interpretation to achieve higher performance in 3D IoU and translation offset metrics. Another possible future research direction includes the use of the NVIDIA TensorRT library [47] to further accelerate the model and increase its throughput.

Acknowledgments

References

[1] David G Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 11 2004.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[3] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6D Object Pose Estimation Using 3D Object Coordinates. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 536–551, Cham, 2014. Springer International Publishing.

[4] Nuno Pereira and Lu'A Alexandre. MaskedFusion: Mask-based 6D Object Pose Estimation. *arXiv e-prints*, page arXiv:1911.07771, 11 2019.

[5] Keunhong Park, Arsalan Mousavian, Yu Xiang, and Dieter Fox. LatentFusion: End-to-End Differentiable Reconstruction and Rendering for Unseen Object Pose Estimation. *arXiv e-prints*, page arXiv:1912.00416, 12 2019.

[6] Meng Tian, Liang Pan, Jr Ang Marcelo H, and Gim Hee Lee. Robust 6D Object Pose Estimation by Learning RGB-D Features. *arXiv e-prints*, page arXiv:2003.00188, 2 2020.

[7] Caner Sahin and Tae-Kyun Kim. Category-level 6D Object Pose Recovery in Depth Images. *arXiv e-prints*, page arXiv:1808.00255, 8 2018.

[8] David Joseph Tan, Nassir Navab, and Federico Tombari. 6D Object Pose Estimation with Depth Images: A Seamless Approach for Robotic Interaction and Augmented Reality. *arXiv*, 2017.

[9] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *arXiv e-prints*, page arXiv:1612.00593, 12 2016.

[10] Zelin Xu, Ke Chen, and Kui Jia. W-PoseNet: Dense Correspondence Regularized Pixel Pair Pose Regression. *arXiv e-prints*, page arXiv:1912.11888, 12 2019.

[11] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Mart\'\in-Mart\'\in, Cewu Lu, Li Fei-Fei, and Silvio Savarese. DenseFusion: 6D Object Pose Estimation by Iterative Dense Fusion. *arXiv e-prints*, page arXiv:1901.04780, 1 2019.

[12] Ge Gao, Mikko Lauri, Yulong Wang, Xiaolin Hu, Jianwei Zhang, and Simone Frintrop. 6D Object Pose Regression via Supervised Learning on Point Clouds. *arXiv e-prints*, page arXiv:2001.08942, 1 2020.

[13] Wei Chen, Xi Jia, Hyung Jin Chang, Jinming Duan, and Ales Leonardis. G2L-Net: Global to Local Network for Real-time 6D Pose Estimation with Embedding Vector Features. *arXiv e-prints*, page arXiv:2003.11089, 3 2020.

[14] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes. In Kyoung Mu Lee, Yasuyuki Matsushita, James M Rehg, and Zhanyi Hu, editors, *Computer Vision – ACCV 2012*, pages 548–562, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[15] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. *arXiv e-prints*, page arXiv:1711.00199, 11 2017.

[16] Tomas Hodan, Pavel Haluza, Stepan Obdrzalek, Jiri Matas, Manolis Lourakis, and Xenophon Zabulis. T-LESS: An RGB-D Dataset for 6D Pose Estimation of Texture-less Objects. *arXiv e-prints*, page arXiv:1701.05498, 1 2017.

[17] He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J Guibas. Normalized Object Coordinate Space for Category-Level 6D Object Pose and Size Estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2019.

[18] Shubham Tulsiani and Jitendra Malik. Viewpoints and Keypoints. *arXiv e-prints*, page arXiv:1411.6067, 11 2014.

[19] Hao Su, Charles R Qi, Yangyan Li, and Leonidas Guibas. Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views. *arXiv e-prints*, page arXiv:1505.05641, 5 2015.

[20] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again. *arXiv e-prints*, page arXiv:1711.10006, 11 2017.

[21] Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker, and Rudolph Triebel. Implicit 3D Orientation Learning for 6D Object Detection from RGB Images. *arXiv e-prints*, page arXiv:1902.01275, 2 2019.

[22] Zhe Cao, Yaser Sheikh, and Natasha Kholgade Banerjee. Real-time scalable 6DOF pose estimation for textureless objects. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2441–2448. IEEE, 5 2016.

[23] Menglong Zhu, Konstantinos G Derpanis, Yinfei Yang, Samarth Brahmbhatt, Mabel Zhang, Cody Phillips, Matthieu Lecce, and Kostas Daniilidis. Single image 3D object detection and pose estimation for grasping. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3936–3943. IEEE, 5 2014.

[24] Chunhui Gu and Xiaofeng Ren. Discriminative Mixture-of-Templates for Viewpoint Classification. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 408–421, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[25] R Rios-Cabrera and T Tuytelaars. Discriminatively Trained Templates for 3D Object Detection: A Real Time Scalable Approach. In *2013 IEEE International Conference on Computer Vision*, pages 2048–2055, 2013.

[26] D P Huttenlocher, G A Klanderman, and W J Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[27] S Hinterstoisser, S Holzer, C Cagniart, S Ilic, K Konolige, N Navab, and V Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *2011 International Conference on Computer Vision*, pages 858–865, 2011.

[28] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes. In Kyoung Mu Lee, Yasuyuki Matsushita, James M Rehg, and Zhanyi Hu, editors, *Computer Vision – ACCV 2012*, pages 548–562, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[29] S Hinterstoisser, C Cagniart, S Ilic, P Sturm, N Navab, P Fua, and V Lepetit. Gradient Response Maps for Real-Time Detection of Textureless Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):876–888, 2012.

[30] D G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157, 1999.

[31] Alvaro Collet, Manuel Martinez, and Siddhartha S Srinivasa. The MOPED framework: Object recognition and pose estimation for manipulation. *The International Journal of Robotics Research*, 30(10):1284–1306, 2011.

[32] Fred Rothganger, Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. 3D Object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints. *International Journal of Computer Vision*, 66:2006, 2006.

[33] Catherine Capellen, Max Schwarz, and Sven Behnke. ConvPoseCNN: Dense Convolutional 6D Object Pose Estimation. *arXiv e-prints*, page arXiv:1912.07333, 12 2019.

[34] Yinlin Hu, Joachim Hugonot, Pascal Fua, and Mathieu Salzmann. Segmentation-driven 6D Object Pose Estimation. *arXiv e-prints*, page arXiv:1812.02541, 12 2018.

[35] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. DPOD: 6D Pose Object Detector and Refiner. *arXiv e-prints*, page arXiv:1902.11020, 2 2019.

[36] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects. *arXiv e-prints*, page arXiv:1809.10790, 9 2018.

[37] Markus Oberweger, Mahdi Rad, and Vincent Lepetit. Making Deep Heatmaps Robust to Partial Occlusions for 3D Object Pose Estimation. *arXiv e-prints*, page arXiv:1804.03959, 4 2018.

[38] Sida Peng, Xiaowei Zhou, Yuan Liu, Haotong Lin, Qixing Huang, and Hujun Bao. PVNet: Pixel-wise Voting Network for 6DoF Object Pose Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page arXiv:1812.11788, 12 2020.

[39] Omid Hosseini Jafari, Siva Karthik Mustikovela, Karl Pertsch, Eric Brachmann, and Carsten Rother. iPose: Instance-Aware 6D Pose Estimation of Partly Occluded Objects. *arXiv e-prints*, page arXiv:1712.01924, 12 2017.

[40] Zhigang Li, Gu Wang, and Xiangyang Ji. CDPN: Coordinates-Based Disentangled Pose Network for Real-Time RGB-Based 6-DoF Object Pose Estimation. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7677–7686, 2019.

[41] Ryosuke Okuta, Y Unno, Daisuke Nishino, S Hido, and Crissman. CuPy : A NumPy-Compatible Library for NVIDIA GPU Calculations. 2017.

[42] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. *arXiv e-prints*, page arXiv:1708.02002, 8 2017.

[43] Gideon Billings and Matthew Johnson-Roberson. SilhoNet: An RGB Method for 6D Object Pose Estimation. *arXiv e-prints*, page arXiv:1809.06893, 9 2018.

[44] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H Wallach, H Larochelle, A Beygelzimer, F d\textquotesingle Alché-Buc, E Fox, and R Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[45] W A Falcon and .al. PyTorch Lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

[46] Pavel Yakubovskiy. Segmentation Models Pytorch. \url{https://github.com/qubvel/segmentation_models.pytorch}, 2020.

[47] NVIDIA. NVIDIA TensorRT, 4 2021.