DeepRob Final Project Report: Effect of Feed-Forward Features in the Performance of PoseCNN

1st Ruiyang Wang
Robotics Department
University of Michigan
Ann Arbor, the United States
ruiyangw@umich.edu

***Abstract*—In this project, we have reproduced the result of PoseCNN in image segmentation and 6D pose estimation. We further investigate how the feed-forward features in PoseCNN network affect its final performance in pose estimation. We found that with our modification of feeding one more feature forward, the 5°5cm accuracy of the PoseCNN network improves from 0.5362 to 0.6414 after training on the PROPS Classification or Detection dataset.**

***Index Terms*—Image Segmentation, 6D Pose Estimation, PoseCNN**

I. Introduction

Pose estimation is a key task in robotic perception, which is the process of extracting and interpreting information from sensor data to understand the robot's environment. Accurate perception is essential for robots to perform a wide range of tasks, from simple manipulation and navigation to more complex tasks such as assembly, inspection, and human-robot interaction.

In the context of robotic manipulation, pose estimation is critical for tasks such as grasping, where the robot must accurately locate and grasp an object. In addition, pose estimation is important for tasks such as assembly, where the robot must align and connect multiple parts together. Accurate pose estimation is also essential for tasks such as inspection, where the robot must detect and identify defects or anomalies in a product.

This project focuses on replicating and improving the performance of PoseCNN [1]. PoseCNN is a deep learning-based approach for 6D pose estimation, which involves determining the position and orientation of an object in 3D space relative to a camera. This approach is important because it has demonstrated state-of-the-art performance in various challenging scenarios, such as cluttered environments and occlusions.

One of the key advantages of PoseCNN is its ability to handle partial occlusions, where parts of the object are obscured from the camera's view. This is achieved by using a multi-scale convolutional neural network that can extract features from different levels of the image pyramid, which we call feed-forward features, allowing the algorithm to detect and match features even when parts of the object are occluded. Our project studies particularly how different features extracted and fed forward can affect the final performance of PoseCNN.

The rest of the paper is structured as follows. Section II will review other approaches in pose estimation, and Section III will briefly introduce the structure and implementation of the PoseCNN framework and our algorithmic extension to improve its performance in extracting more feed-forward features. Section IV will present simulation results, and Section V concludes our findings.

II. Related Work

Traditional methods for pose estimation usually involve feature extraction and matching techniques. One of the most widely used feature extraction techniques is the Scale-Invariant Feature Transform (SIFT) algorithm, which is used to extract key points and descriptors from images [2]. Other traditional methods for pose estimation include the iterative closest point (ICP) algorithm, which is used to align 3D point clouds, and the perspective-n-point (PnP) algorithm, which is used to estimate the pose of an object from its 3D model and 2D image projections [3].

Deep learning-based approaches for 6D pose estimation have been developed in response to the limitations of traditional pose estimation methods. Some of the other deep learning-based approaches for 6D pose estimation include DeepIM [4], PVNet [5], and DenseFusion [6].

DeepIM is a method that uses a convolutional neural network to estimate the 2D bounding box of the object, which is then used to refine the 6D pose estimation. PVNet is a method that uses a multi-stage network to estimate the 2D projection of the object in the image, which is then used to estimate the 3D pose. DenseFusion is a method that uses a dense fusion network to estimate the 3D pose of the object, which takes into account both appearance and geometric information.

Compared to these methods, PoseCNN has the advantage of being able to handle partial occlusions and reflective surfaces, which can be challenging for traditional methods and some other deep learning-based methods.

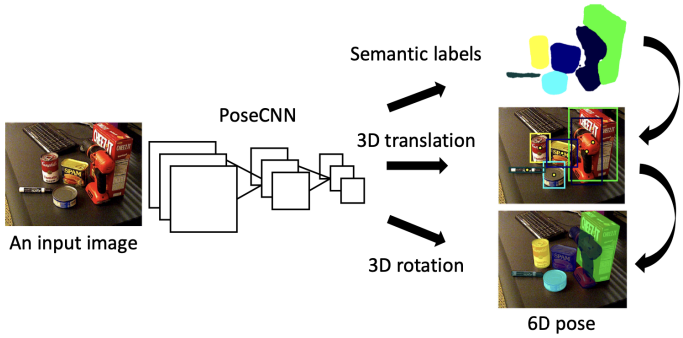

Fig. 1: Illustration of the problem that PoseCNN tries to solve

In summary, PoseCNN stands out for its ability to handle challenging scenarios such as occlusions and reflective surfaces, while still maintaining efficiency and real-time performance. It has been evaluated and compared to other methods in academic papers, including the original paper by Peng et al [5]

III. Methodology and Implementation

A. Problem Definition

The problem we are trying to solve is, given an input image, we want to identify objects contained inside the image, label their positions in pixels, and correctly estimate their pose with respect to the camera. This process can be better illustrated with a flowchart from PoseCNN paper in Fig. 1.

B. PoseCNN Network

PoseCNN network tries to solve the aforementioned pose estimation problem using a deep-learning neural network. The overall structure of the PoseCNN network is summarized in Fig. 2.

We implemented a variant of the illustrated PoseCNN network because of our limited number of training samples and the capability of free computing resources from Google CoLab. We used the first 30 layers from the pre-trained VGG-16 model to extract features from the RGB images rather than training the feature extractor from scratch. Everything else stays the same as shown in Fig. 2

C. Algorithmic Extentsion

As shown in Fig. 2, the default feature extractor from PoseCNN outputs two features and feeds them to later layers in image segmentation branch and translation branch. In our implementation, the default feature extractor outputs features after the input data has been passed through the 23rd layer and the 30th layer. Our project aims to investigate how the different extracted features can affect the final performance of PoseCNN.

The modification we implemented is that we feed three features instead of feeding two features to future branches.

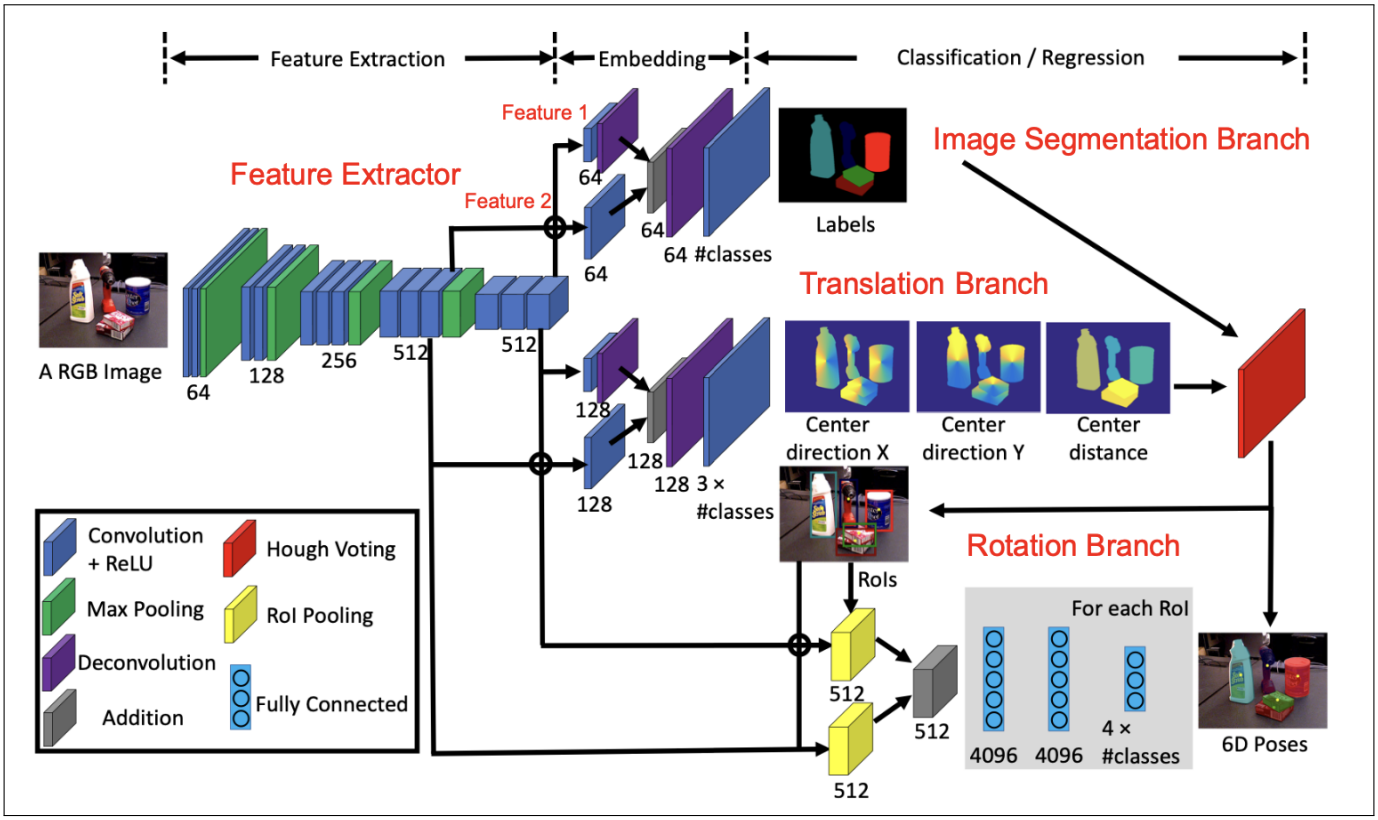

Fig. 2: Architecture of PoseCNN for 6D object pose estimation.

Our feature extractor outputs a new feature after the input data passes through the 15th layer in addition to the two features outputted from the default feature extractor from PoseCNN.

The intuition behind this modification is that features outputted from earlier layers will include more detailed information about the original data input, thus might improve the performance of the network in image segmentation. As a result of that, it might improve the models' performance in pose estimation later on. Meanwhile, more detail might lead to an increase in the complexity of the model, thus making the model take longer and more resources to train.

We also implemented a variant of PoseCNN by having the feature extractor output only its final feature after passing through all layers for baseline comparison.

IV. Experiments and Results

In our experiment, we used the GPU provided by Google CoLab as our hardware for both training and evaluating models on the PROPS Classification or Detection dataset. We evaluated three variants of the PoseCNN model in simulation: PoseCNN with 1 extracted feature (Baseline Comparison), PoseCNN with 2 extracted features (The Original PoseCNN published), and PoseCNN with 3 extracted features (Our modification).

For the image segmentation task only, we have trained all models with 500 images from the PROPS dataset for 3 epochs with a batch size of 4. The first set of qualitative results is shown in Fig. 3. This is similar to the performance shown in the original PoseCNN paper. As we can see from the figure, the predicted segmentation images from PoseCNN models become finer when we increase the number of features extracted. This result matches our expectation that if we include more features from earlier layers, which comprise more details of the RGB input image, we will have finer grid performance in the segmentation task. With this result, we also expect that PoseCNN will have better results with our modification of extracting 3 features from the feature extractor comparing to the orginal published PoseCNN with 2 extracted features.

For 6D pose estimation, we have trained all models with 500 images from the PROPS dataset for 10 epoch with batch size of 4, and validated the models using 500 images from the PROPS dataset. The quantitative results are illustrated in Fig. 4. As we can see from the figure, all variant of PoseCNN models produce reasonable results after the training.

The quantitative performance of the three variants of PoseCNN is summarized in Table. I. We have used the metric of 5°5cm accuracy, in which a correct prediction is defined as one with a rotation error of less than 5° and a translation error of less than 5cm.

The result suggested that all models take about the same amount of time to train, and their training losses are very close to each other. Meanwhile, the PoseCNN with 2 extracted features seems to outperform both the variant model with 1 and with 3 extracted features in 5°5cm accuracy

We think this is because the model with 3 extracted features might need more epochs to train compared to the other two

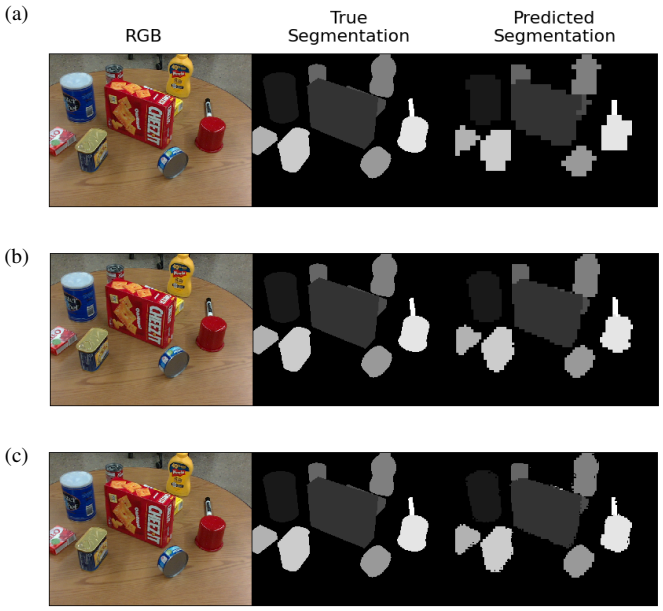

Fig. 3: Segmentation Qualitative Performance for PoseCNN variants with (a) 1 extracted feature, (b) 2 extracted features, (c) 3 extracted features.

Model	# of Extracted Features	Training Time (s)	Training Loss	Validation 5°5cm Accuracy
PoseCNN	1	1280	0.0972	0.4636
	2	1312	0.1031	0.5362
	3	1283	0.0859	0.4366

TABLE I: Simulation result comparison between PoseCNN model with extracted 1, 2, and 3 number of features after training for 10 epochs.

models because of its complexity. So we doubled the number of epochs to train. The qualitative result is shown in Fig. 5, and the quantitative results are summarized in Table. II

This set of results matches our expectations because our modification by adding another feed-forward feature from the extractor outperforms the original published PoseCNN network in validation 5°5cm accuracy. In addition to that, our modification doesn't take significantly longer to train. With more training epochs, our performance in 5°5cm Accuracy

Model	# of Extracted Features	Training Time (s)	Training Loss	Validation 5°5cm Accuracy
PoseCNN	1	2683	0.0772	0.6383
	2	2617	0.0831	0.4957
	3	2752	0.0752	0.6414

TABLE II: Simulation result comparison between PoseCNN model with extracted 1, 2, and 3 number of features after training for 20 epochs.

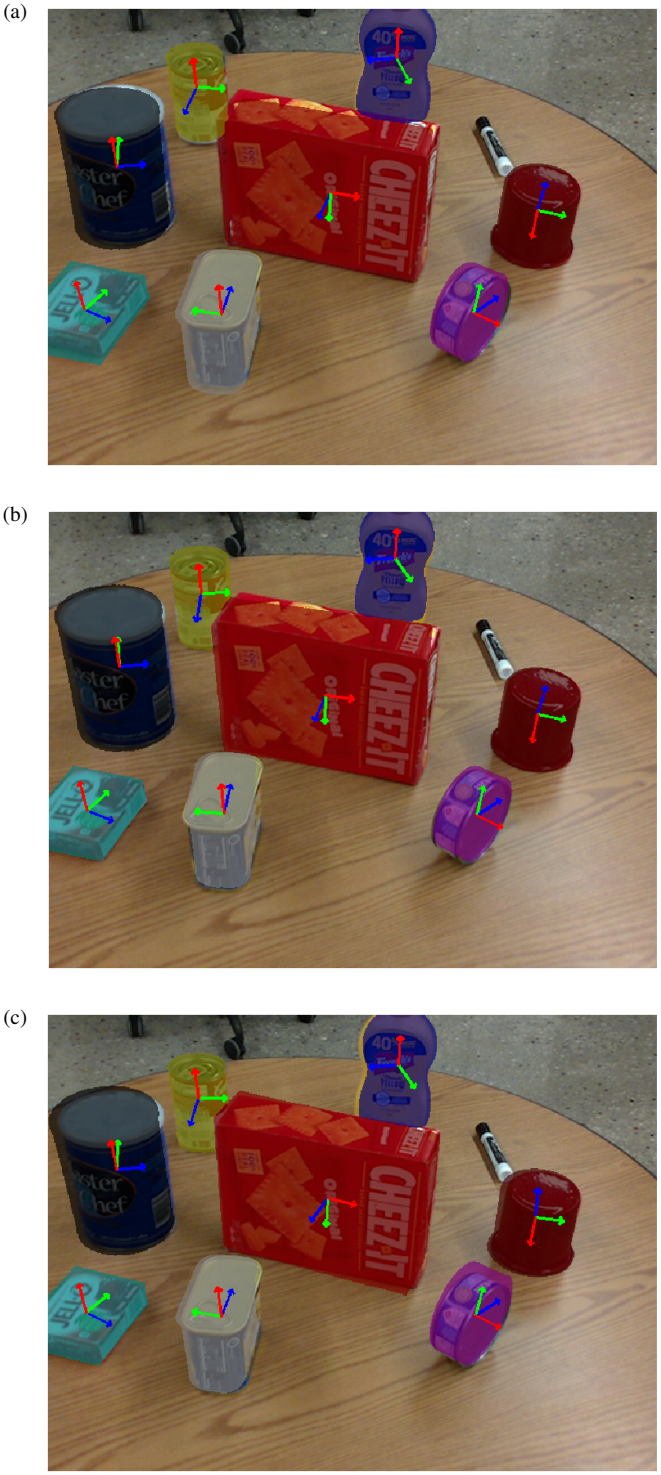

Fig. 4: Pose Estimation Qualitative Performance for PoseCNN variants with (a) 1 extracted feature, (b) 2 extracted features, (c) 3 extracted features after training for 10 epochs.

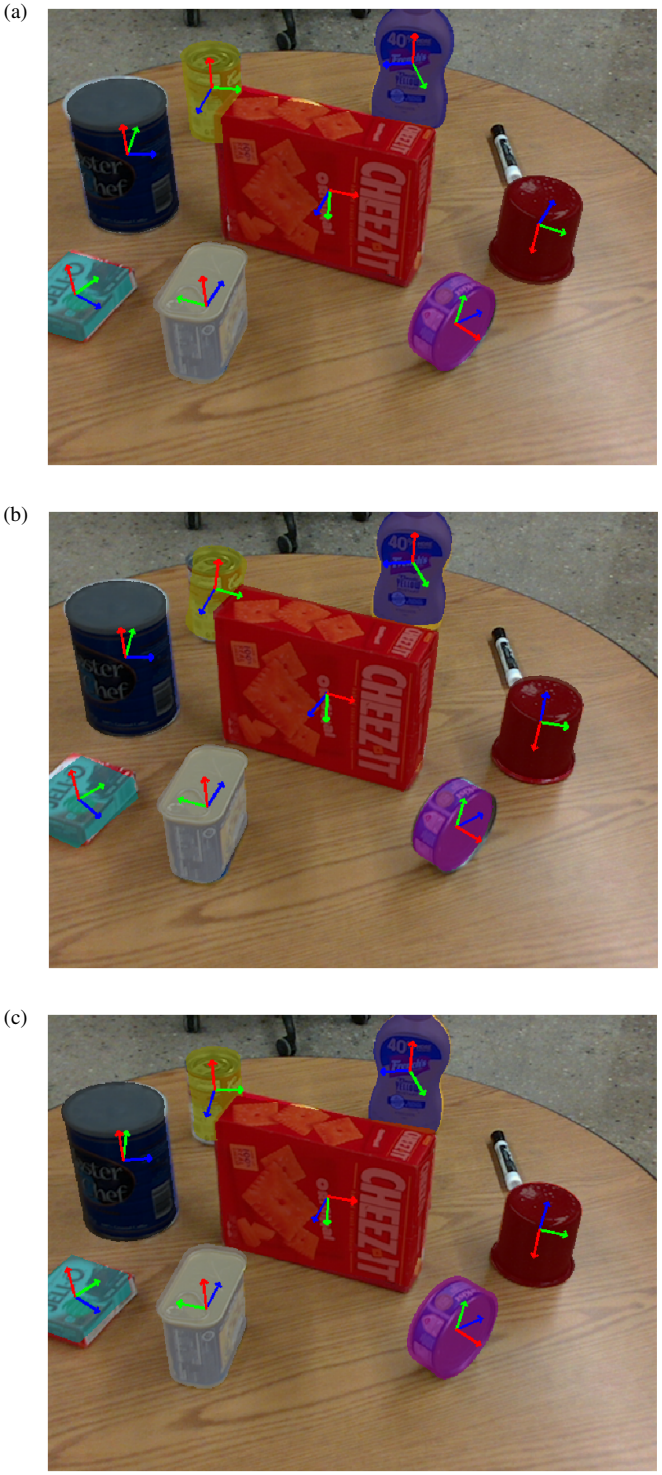

Fig. 5: Pose Estimation Qualitative Performance for PoseCNN variants with (a) 1 extracted feature, (b) 2 extracted features, (c) 3 extracted features after training for 20 epochs

improves from 0.5362, which is the best performance using PoseCNN with 2 feed-forward features, to 0.6414, which is the best performance achieved with our modification. We also believe the performance of PoseCNN model with 3 extracted features might perform even better if we have more data to train.

V. Conclusions

In summary, this project investigated how the feed-forward features in the PoseCNN models affect the final performance in image segmentation and pose estimation.

We found that with additional features outputted from earlier layers of the network, the network can produce image segmentation with a finer grid. This is because features from earlier layers comprise more detailed information about the input data, thus helping reconstruct a finer segmentation map in pixels. For the same reason, we think that our modification of feeding an extra feature outputted from early layers forward can improve the performance of PoseCNN in pose estimation.

We first reproduced the result from PoseCNN paper using PROPS Classification or Detection dataset 6D pose estimation, and evaluated variant of PoseCNN models with different number extracted features. We found that with a small number of training epochs, adding extra features will not improve the performance of the model in pose estimation. We think this is because we have a more complicated model after adding more feed-forward features, thus we need to train it for more epochs.

As expected, if we increase the number of training epochs, adding extra feeding forward feature does improve the model's performance in 5°5cm Accuracy from 0.5362 to 0.6414, which is the best performance we have seen across all variants of PoseCNN we implemented.

We also believe that if we have more training data, our variant of PoseCNN with 3 feed-forward features will perform even better because of its capability of containing more details about the input image.

References

[1] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," *arXiv preprint arXiv:1711.00199*, 2017.

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91–110, 2004.

[3] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[4] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "Deepim: Deep iterative matching for 6d pose estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 683–698.

[5] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao, "Pvnet: Pixel-wise voting network for 6dof pose estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4561–4570.

[6] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese, "Densefusion: 6d object pose estimation by iterative dense fusion," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 3343–3352.