CSE 275 HW3: UNet + DenseFusion + ICP for 6D Pose Estimation

Arth Shukla
University of California, San Diego
arshukla@ucsd.edu

1. Method Overview

1.1. Learning Method (DenseFusion + ICP Refinement)

Given an image and its corresponding metadata, we first perform semantic segmentation on the image with a UNet [2]. Next, for each segmented object, we get a cropped RGB image and a point cloud lifted from the depth image. Then, we pass the point cloud and cropped image to a DenseFusion network to predict per-pixel poses and corresponding confidence values [3]. Next, we select the pose with highest confidence. Finally, we use the DenseFusion predicted pose as a starting pose for the ICP algorithm to further refine the predicted pose [1].

We train the DenseFusion network in **two stages** based on minimizing error for objects with infinite order symmetries. The first stage minimizes RRE symmetry for symmetric objects and the second stage further finetunes on other objects. Additional details are provided in Sec. 3 and Sec. 5.2.

This method performs well, resulting in a model which acheives >90% accuracy on HW2's test set and >80% accuracy on HW3's test set. Furthermore, this model is fast at inference if given a GPU, with and without ICP refinement, allowing for real-time pose estimation.

1.2. Algorithmic Method (Standalone ICP)

While ICP performs well if given a good initialization, finding this initialization is difficult, especially with heavy occlusions. To alleviate this, when using only ICP, we will initialize ICP randomly many times and pick the best result. Additional details on random initialization are available in Sec. 5.4.

Furthermore, we would like to run ICP to move the object model to the depth-lifted point cloud. However, the depth-lifted point cloud has large occlusions since the camera can only see part of the object. So, we must run ICP to find the transformation $[R^* \mid t^*]$ which moves the depth-lifted point cloud to match the model. Then, the estimated pose will be the opposite, $[R \mid t] = [{R^*}^T \mid -t^*]$.

1.3. Model Variations

1. UNet + Standalone DenseFusion
2. UNet + DenseFusion + ICP Refinement
3. UNet + Standalone ICP (vanilla ICP with many random initializations)

2. Network Architecture

The network architecture is pictured in full in Fig. 1.

2.1. Semantic Segmentation

We perform semantic segmentation on the RGB image using a UNet with bilinear interpolation [2]. Since there is at most one instance of each object in the image, semantic segmenation is sufficient. This segmentation result is applied to both the RGB image and the depth image.

Following the original DenseFusion paper, we use OpenCV to find a bounding box for the object and crop the image. Furthermore, we use the camera intrinsic and extrinsic to lift a pointcloud from the depth image. Additional data processing information is available in Sec. 4.

2.2. DenseFusion

The network architecture is pictured in full in Fig. 1. We implement the same architecture as the original DenseFusion paper [3].

We use PSPNet with ResNet-18 backend for color embeddings (code borrowed from `https://github.com/Lextal/pspnet-pytorch` with permission from Professor) [4]. We use a simplified PointNet with Average Pooling to obtain geometric features and a global feature. We concatenate the color embeddings, geometric features, and global feature to get per-pixel embeddings.

We regress rotations over the full quaternion space (4-dim outuput), and each quaternion is converted to a rotation matrix in SO_3 before loss calculation. Translations are 3-dim output, and confidence values are 1-dim output.

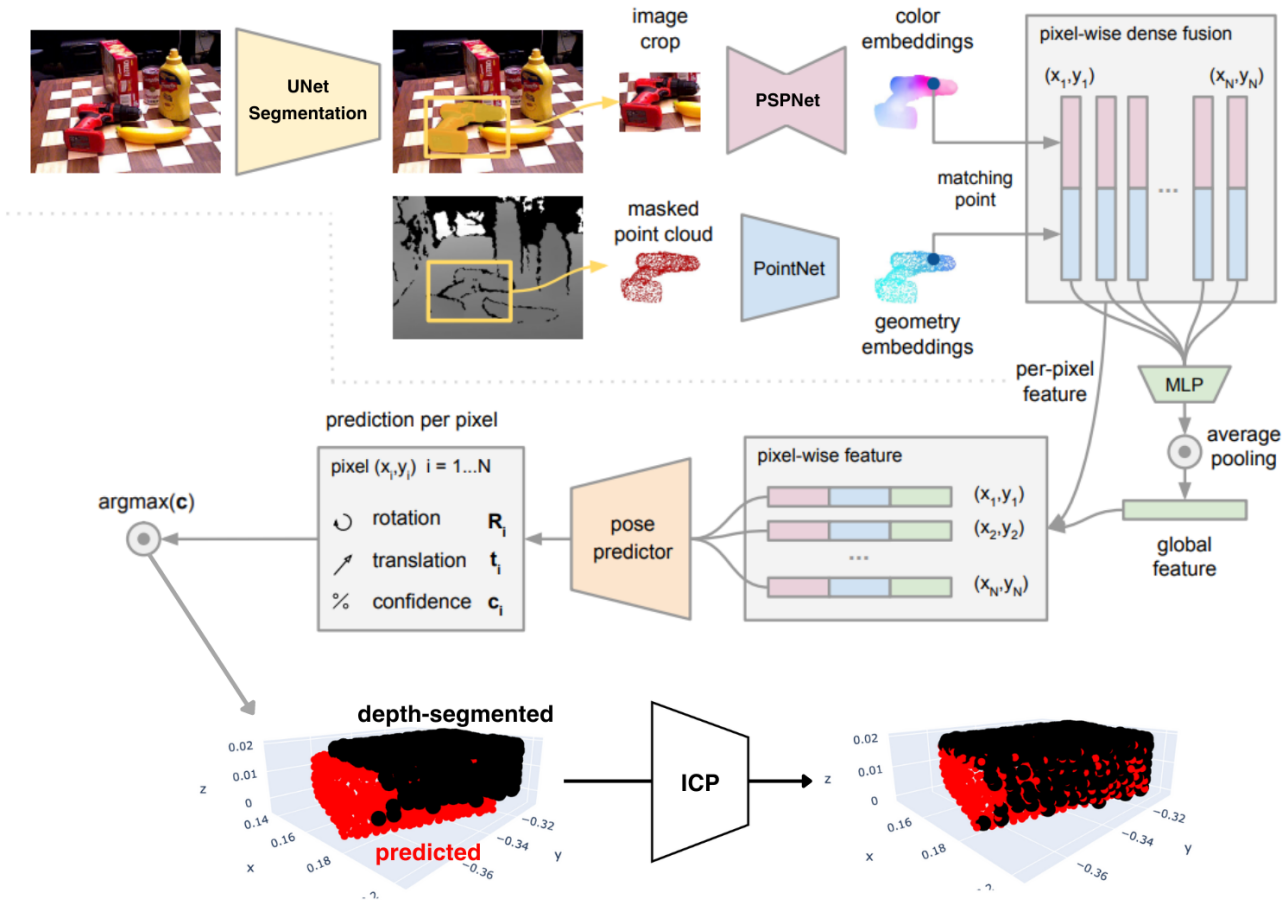


Figure 1. Full UNet + DenseFusion + ICP Refinement architecture. The top part of the image is altered from the original DenseFusion paper [3]. Note that the pre-refinement prediction shown in the figure was manually altered to make the difference more visibly noticeable. In reality, the difference is smaller, though refinement still results in improvements.

2.3. ICP Refinement

While the network can predict good pose estimations, sometimes the predictions have small rotational or translation error. To minimize these errors, we can refine the network's predicted pose by running ICP using the predicted pose as a starting point. We assume the network provides good pose estimations with low rotation and translation error. Thus, ICP should only need to minimally alter the predicted transformation to achieve a better pose estimate.

3. Losses

For our UNet, we use a simple cross-entropy loss. For Densefusion, we combine multiple shape- and symmetry-aware pose estimation losses.

First, our loss must take into account the symmetries of the objects. Second, our loss must apply to each per-pixel pose prediction, and it must also take into account the predicted confidence values for each per-pixel pose prediction.

We are given a set of M points x, with jth point x_j. We denote ground truth pose $p = [R\,|\,t]$. For objects with n-order of symmetries, we have multiple ground-truth rotations $\mathcal{R} = \{R^1, \ldots, R^n\}$. For a dense-fused pixel i, we define predicted pixel pose $[\hat{R}_i\,|\,\hat{t}_i]$. We now formulate symmetry-aware loss L_i^P. First, we separate objects into three categories: no symmetries, finite order of symmetries, and infinite order of symmetry.

For objects with no symmetries, we use a simple per-point MSE loss and average over all dense-fused pixels:

$$L_i^P = \frac{1}{M} \sum \| (Rx_j + t) - (\hat{R}_i x_j + \hat{t}_i) \|$$

For objects with finite order of symmetries, we use a min-of-N loss:

$$L_i^P = \min_{R^k \in \mathcal{R}} \frac{1}{M} \sum \| (R^k x_j + t) - (\hat{R}_i x_j + \hat{t}_i) \|$$

Finally, for objects with infinite order of symmetries, we

use Chamfer loss:

$$L_i^P = \frac{1}{M} \sum \min_{k \in \{1,\dots,M\}} \|(Rx_j + t) - (\hat{R}_i x_k + \hat{t}_i)\|$$

However, we note that Chamfer loss can often get stuck in suboptimal local minima. So, to prevent this, we add a hyperparameter `min_over_cham` which is between 0 and 1. This defines the probability that we will use min of N loss over Chamfer loss for objects with infinite symmetry. Intuitively, if our network gets stuck in some local minima with Chamfer loss, introducing min of N loss could help our network to leave those local minima while minimizing the number of otherwise valid pose estimations the altered loss penalizes.

For this reason, we **split training into two stages**: during the first stage, we have nonzero `min_over_cham`, and during the second stage, we set `min_over_cham=0`. Specifics are discussed in Sec. 5.2, and we show in Sec. 6 that this improves performance by decreasing rotation symmetry error.

Finally, similar to the original DenseFusion paper [3], we weigh per-pixel loss by predicted confidence values and add confidence regularization term w to get per-data point loss:

$$L = \frac{1}{N} \sum_i \left(L_i^P c_i - w \log(c_i) \right)$$

4. Data Processing

To increase the speed of training, we process our data ahead of time and save all relevant processed data as files. This way, we need only read files when loading data during training.

Note that we use the given train/validation/test splits. Furthermore, to save on storage space and train time, for training and validation data we select only the first 100 scenes for each level.

4.1. UNet Semantic Segmentation Data

Here, we need only save the RGB image and the ground truth segmentation masks. The ground truth segmentation mask has 82 classes, 79 for different objects, and 3 for background, table, or pipe.

4.2. DenseFusion Data

Given an RGB-D image with associated metadata, we perform the following processing:

1. Run UNet on RGB image to get segmentation mask
2. For each object found in segmentation,
   (a) Use segmentation mask to obtain per-object mask
   (b) Use per-object mask to obtain cropped RGB image using OpenCV bounding rect
   (c) Sample up to 1000 pixels randomly from per-object mask, save as a 'choose' mask
   (d) Apply choose mask to depth image, sampling up to 1000 points from depth image pixels. Use inverse intrinsic to convert to points to camera frame, then inverse extrinsic to convert to world frame
   (e) Save object model and metadata like object name and index
   (f) For training data, save ground truth pose. Apply ground truth pose to model to get target

We treat each individual instance of an object as one data point. So, for each data point, we save the cropped RGB, 'choose' mask, depth-lifted point cloud, object model, and metadata. For training/validation data, we also save the ground truth pose and target.

5. Training / Algorithm Running Details

The DL models are implemented in PyTorch and we use Adam optimizer throughout (with default hyperparameters aside from learning rate).

5.1. UNet

We use learning rate of 10^{-4}, batch size of 1, and we train for 4 epochs. In ablations, we train ablate on up-sampling method: Transpose Convolution or Bi-Linear Interpolation. Additionally, for comparison we train a vanilla SegNet, the segmentation model used by the original DenseFusion paper.

5.2. DenseFusion

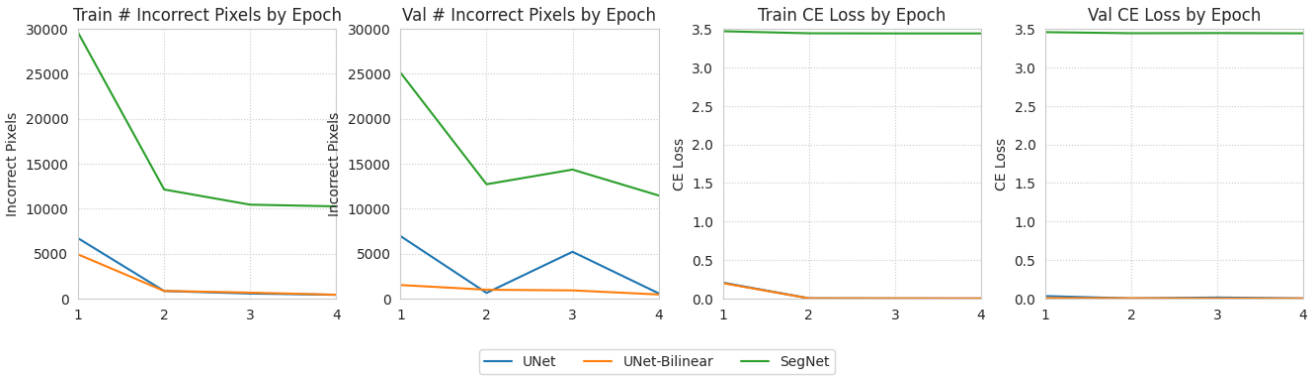
For all experiments, we use learning rate of 10^{-4}, batch size of 36, confidence regularization $w = 0.015$, and we train for 300 epochs.

As noted in Sec. 3, we split DenseFusion training into **two stages**: in stage 1, we have nonzero `min_over_cham` probability, and in stage 2 we set `min_over_cham=0`. We switch from stage 1 to stage 2 once we reach train accuracy of 60%.

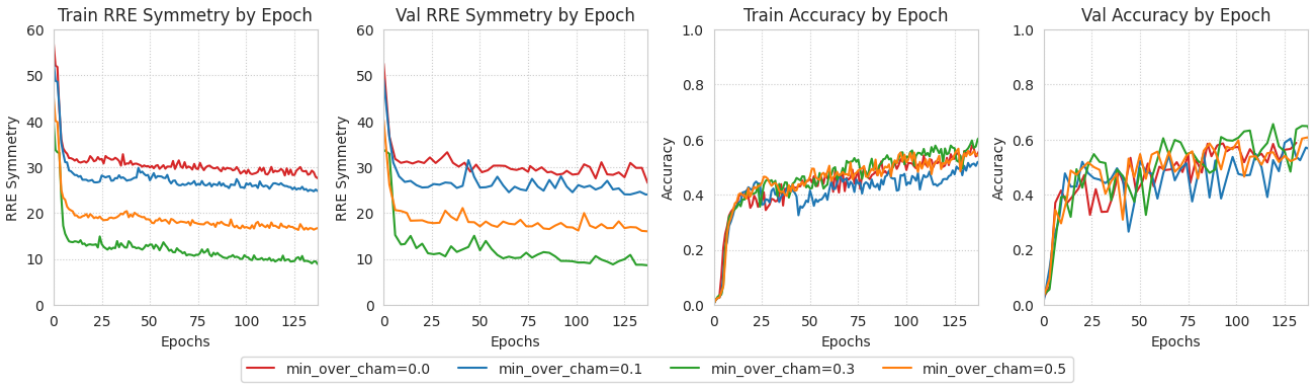
In ablations, we train on `min_over_cham` values of 0, 0.1, 0.3, and 0.5 for stage 1 and values of 0 and 0.3 for stage 2. Per our ablation results, our optimal model has `min_over_cham` value of 0.3 during stage 1 and value of 0 during stage 2.

5.3. ICP Refinement

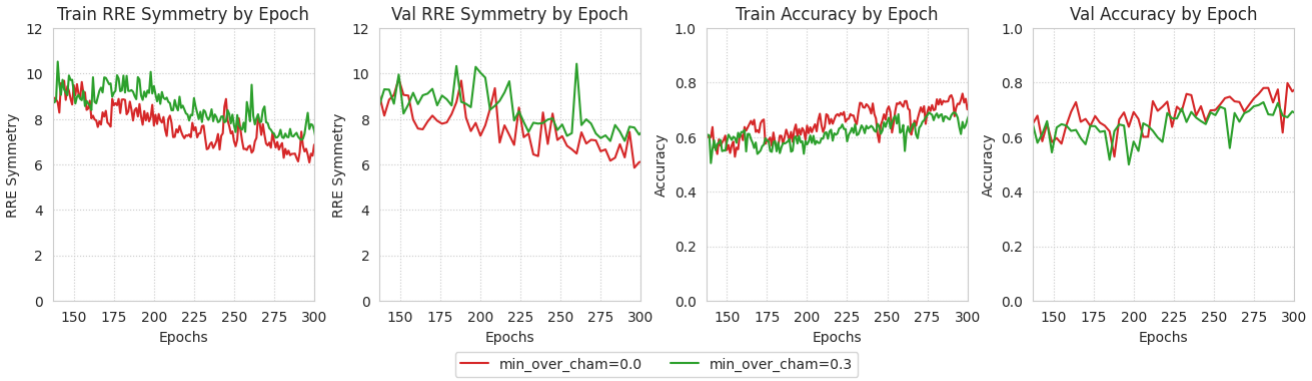
During ICP refinement, we use DenseFusion predicted pose as starting pose and run only one attempt. The algorithm refinement attempt is given a maximum of 1000 steps.



(a) Ablations for Semantic Segmentation Networks: UNet, UNet Bilinear, SegNet. Note that each image is 1280x720, totaling 921,600 pixels.



(b) Ablations for DenseFusion Stage 1 training. Training stops at epoch 137, when the min-over-cham=0.3 model achieves >60% train accuracy.



(c) Ablations for DenseFusion Stage 2 training. Both training curves resume from epoch 137 of the min-over-cham=0.3 Stage 1 curve.

Figure 2. Segmentation and DenseFusion training ablations.

If the average distance in points changes by less than 10^{-30}, we return early.

5.4. Standalone ICP

Since standalone ICP is not given a good starting pose, for each attempt of ICP, we initialize the depth-lifted point cloud with a random rotation and translation.

First, we apply the random translation. For each model point cloud, we define the model center as the mean of all points and the model radius as the distance between the center and the further point in the model. Then, we create a ball centered at the model center and with radius 2 times the model radius. We then move the depth-lifted point cloud to random points on this ball by randomly sampling spherical coordinates θ and ϕ.

Next, we apply the random rotation. To do this, we sam-

ple a random 3x3 matrix with values over $[0, 1)$ which we convert to a rotation matrix with SVD decomposition. Finally, we apply this random rotation to the depth-lifted point cloud as well.

An example of this random initialization is seen below in Fig. 3:

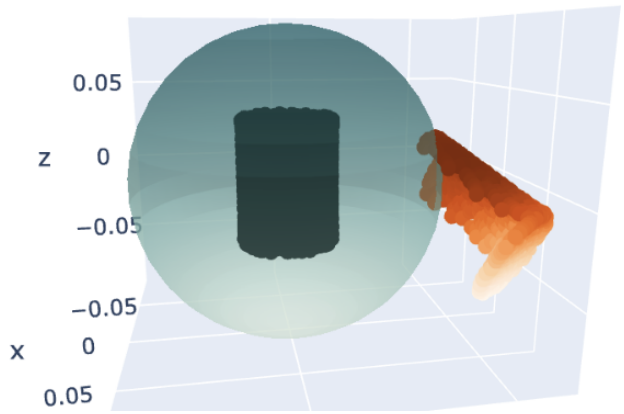


Figure 3. An example of our random initialization strategy. We have a ball centered at the model's center with 2 times model radius. We move our depth cloud to somewhere on the ball, then apply a random rotation before running ICP.

We run versions with 1, 5, and 500 max attempts, 1000 max iters each, and early return if the average distance in points changes by less than 10^{-10}.

6. Ablations

6.1. Semantic Segmentation

As shown in Fig. 2a, we run 4 epochs of training on our segmentation network. We run an ablation on the upsampling method: Transpose Convolution or Bi-Linear Interpolation. We additionally train a SegNet for additional comparison.

We find that UNet with bilinear interpolation performs the best with more stable validation results. Additionally, the SegNet performs significantly worse, dwarfing the UNet CE losses on the chart.

6.2. DenseFusion Stage 1

As shown in Fig. 2b, we run Stage 1 with `min_over_cham` values of 0, 0.1, 0.3, and 0.5. Training for Stage 1 is stopped when `min_over_cham=0.3` achieves >60% train accuracy at epoch 137.

All nonzero values perform better than 0, indicating that Chamfer-only loss for objects with infinite order symmetries is more prone to getting stuck in local minima. The best model ablation is the model which uses `min_over_cham=0.3`, which is able to achieve train and validation RRE symmetry <10 by the end of Stage 1.

This reduction of RRE symmetry is reflected in the accuracy metrics, where the `min_over_cham=0.3` model has the best train and validation accuracies.

6.3. DenseFusion Stage 2

As shown in Fig. 2c, we run Stage 2 with `min_over_cham` values of 0 and 0.3. These runs resume from epoch 137 of the optimal `min_over_cham=0.3` model from Stage 1. The purpose of the Stage 2 ablation is to show that, despite results from Stage 1, running the model with nonzero `min_over_cham` throughout is suboptimal.

The Stage 2 `min_over_cham=0` model outperforms the Stage 2 `min_over_cham=0.3` model. This indicates that, while nonzero `min_over_cham` can reduce the likelihood that our model gets stuck in local minima due to the Chamfer loss during Stage 1, the min over N loss can still hamper overall performance during Stage 2 since it does not take into account infinite order symmetries.

Therefore, the optimal training method is our two-stage method: in the first stage we set `min_over_cham=0.3` and in the second stage we set `min_over_cham=0`.

7. Performance and Visualizations

Model	HW2	Runtime	HW3	Runtime
ICP 1/obj	65.40	59s	60.00	3m10s
ICP 5/obj	90.00	5m50s	81.73	17m16s
ICP 500/obj	**98.13**	9h39m46s	**89.03**	29h36m7s
DenseFusion	81.87	43s	**83.53**	**2m17s**
DF+ICP	**90.73**	**54s**	83.63	3m41s

Table 1. Test set accuracy and runtime for HW2 and HW3. Best accuracy is **bolded** and best accuracy/runtime combination is **bolded pink**.

Tab. 1 contains final accuracy of each model on the test set, as well as runtime for each model. We include metrics for both HW2 and HW3. However, note that for HW2 we use ground-truth segmentation masks, not UNet.

Furthermore, as we use our UNet during data processing, the runtime numbers do not include time to segment and crop each image, lift depth to point cloud, and sample points.

Finally, note that, in all tests, data points are passed one-at-a-time. Therefore, in real-world applications, one could batch all objects from one RGB-D image for increased runtime efficiency.

Accuracy is a percentage and runtime is measured in hours (h), minutes (m), and seconds (s).

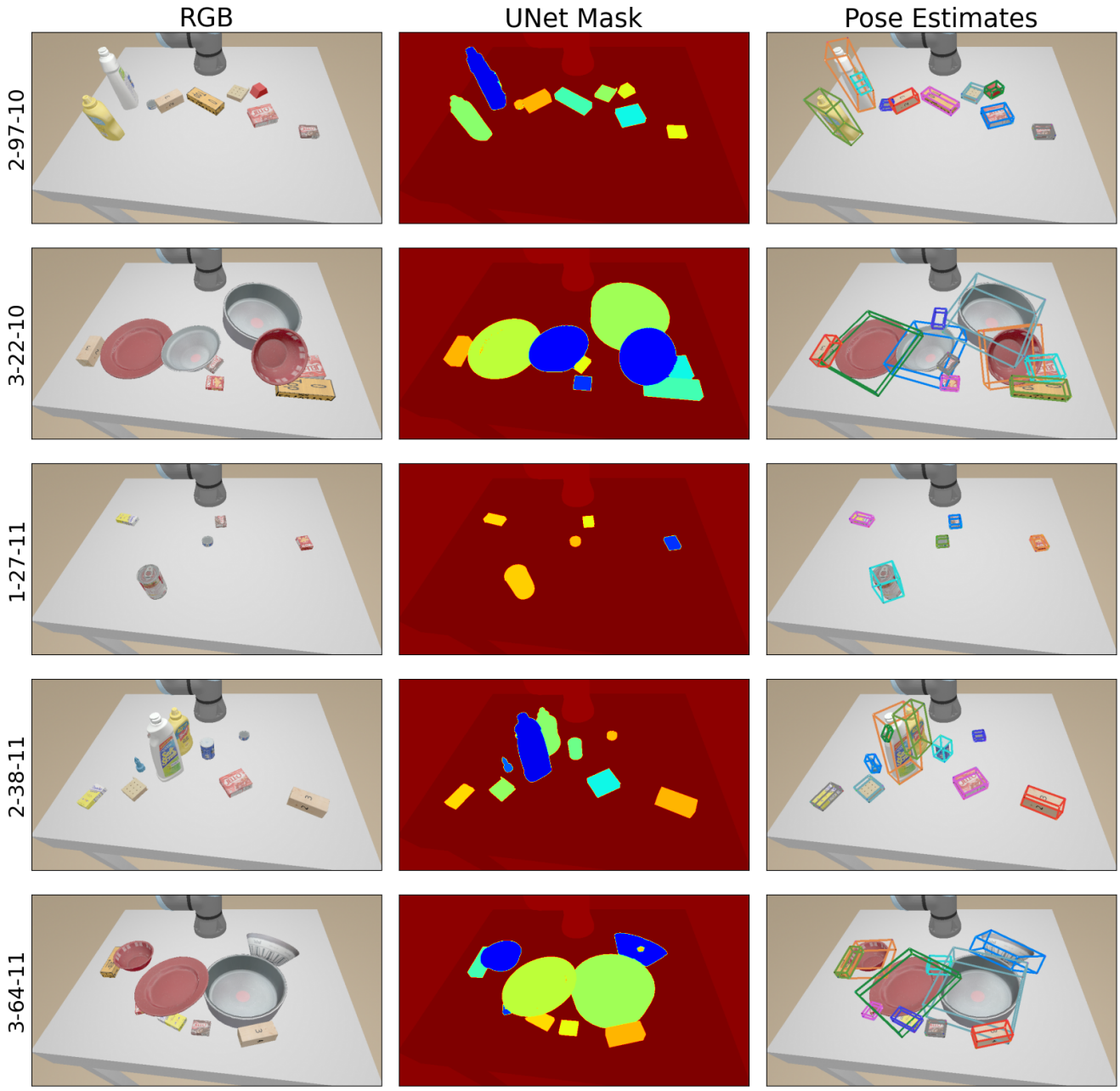


Figure 4. Visualizations of RGB input, UNet segmentation, and Pose Estimation results from DenseFusion + ICP Refinement.

Our DenseFusion models are able to achieve the highest tier of $>80\%$ accuracy on both benchmarks with reasonable computational time. Additionally, our ICP results show that learning-based methods are clearly better when inference speed is important. ICP with 1 attempt per object has similar runtime to our DenseFusion models, but sees a significant drop in accuracy. ICP with 5 attempts per object has similar performance to our DenseFusion models, but about 6-8 times longer inference time.

That being said, our ICP was able to achieve very high accuracy by running 500 attempts per object. However, this is only after running for hours, which is not feasible for real-time pose estimation.

Therefore, we conclude that the optimal model choice is generally DenseFusion + ICP refinement.

8. Future Means of Improvement

To improve accuracy, we can add more data to the model. We only used variants for the first 100 scenes for each level, but there was a lot more data available in the zip file. More data would mean longer train time, but it would also help reduce outlier cases and reduce chances of staying stuck in local minima. We could also provide more train time. The original DenseFusion paper uses 500 epochs [3], however we only use 300.

There are also deep learning-based iterative refinement methods, including one suggested by the original DenseFusion paper [3]. The ICP refinement can only reduce error for estimates that are already "good". For example, if the network predicts with low rotation error but high translation error, the cloud could be oriented away from the predicted model pose. In this case, ICP would not be able to solve in one attempt without additional random initializations. However, a learning-based iterative refinement procedure doesn't strictly require good orientation, and could thus handle these cases more effectively.

References

[1] Paul Besl and H.D. McKay. A method for registration of 3-d shapes. ieee trans pattern anal mach intell. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14:239–256, 03 1992. 1

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. 1

[3] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. Densefusion: 6d object pose estimation by iterative dense fusion, 2019. 1, 2, 3, 7

[4] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network, 2017. 1