Deep Learning Project

Kevin Wang
IX
Imperial College London
kevin.wang25@ic.ac.uk

***Abstract*—Two microrobot perception challenges are addressed during this report, pose and depth estimation of microrobots. Pose estimation is formulated as a 40 class classification problem, recognizing 40 different unique combinations of pitch and roll as poses. On the other hand, depth estimation addresses a regression problem.**

***Index Terms*—microrobot perception, pose classification, depth regression, convolutional neural networks, ResNet18**

I. Introduction

A. Motivation

An important topic in biomedicine, specifically in dynamic microscale environments, is the application of optical microrobots controlled by optical tweezers [1]. Through microrobots, manipulation of micro-scale cells and other biological objects is possible [1]. This requires closed-loop control of microrobots which essentially depends on accurate estimations of pose and depth [1]. However, due to microscopic imaging having issues such as defocusing, optical diffraction and noisy conditions in those environments, it is difficult to reliably estimate pose and depth [1]. This motivates approaches to design learning models that are able to effectively extract feature information from microrobot images and output their pose and depth.

B. Background

In this report we analyze microrobot perception with microscopy images using the OTMR dataset [1]. The dataset contains images of a single type of microrobot with an assigned pitch and roll configurations, and depth. Through this we are able to formulate two supervised learning problems: pose and depth estimation [1].

II. Objectives

This report covers:

1) Preprocess images and map them with their appropriate pose and depth.
2) Train and evaluate CNN models for pose estimation as a 40 class classification problem.
3) Train and evaluate CNN models for depth estimation as a regression problem.
4) Hyperparameter tuning for best performing model, to optimize performance.

III. Methodology

A. Data Loading and Exploration

The dataset structure is composed of a main folder, '2025_Dataset' [1], with subfolders corresponding to a specific microrobot pose with a naming convention P`Pitch`_R`Roll`. Each pose folder contains images of microrobots, and one depth.txt file with depth values that match each image filename. We scan for pose folders by parsing pitch and roll from the folder name, creating a dictionary corresponding to 40 unique poses giving them an index each. We proceed to create a sample list mapping each image with their `image path`, `pitch`, `roll`, a `pose class` index, and the corresponding `depth`. To verify images and labels are loaded correctly, we perform an EDA. Firstly, we observe Figure 1, four different examples of microrobot images are displayed, each with a different pose class and their corresponding depth. Through this we can confirm not only that our dataset is working, but also that each class indeed depicts a different pose.

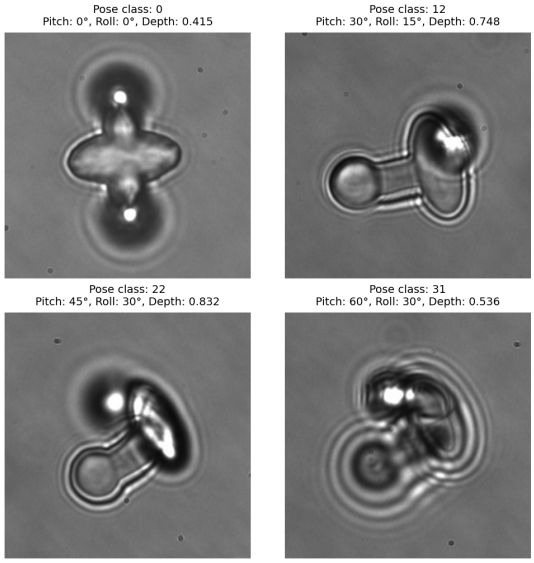

Fig. 1. Sample Image Visualization

We then compute and plot the distribution of the 40 pose classes to assess the class imbalance. As seen in Figure 2, we can clearly observe that the classes are perfectly balanced with each class holding 50 samples for a total of 2000 sample

observations. This cancels the risk of the accuracy being inflated by always predicting the most dominant class.

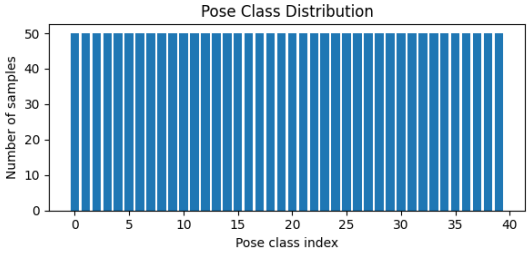

Fig. 2. Pose Class Distribution

In contrast, the depth histogram as shown in Figure 3 is less even, containing different number of samples across all values in its range. A higher concentration can be observed at a lower depth value and at a high depth value approaching 1. On the other hand, a smaller concentration can be seen in the upper middle value range, roughly from 0.6 to 0.8. Although having an uneven distribution, no severe skewness or outlier is observable.

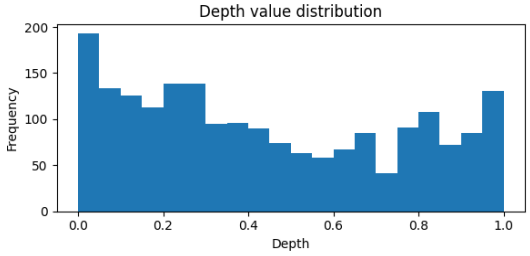

Fig. 3. Depth Value Distribution

B. Data Preprocessing

Moving forward, we prepare the data for model design and application. We decide to split the data into training, validation and test sets using a 60/20/20 ratio with stratification on pose class index to ensure each split maintains equal class distribution. This is completed before augmentation and normalization in order to prevent data leakage [2]. Subsequently, a very weak augmentation is applied to the samples in the training set using `ColorJitter` with brightness and contrast of 0.1 to simulate minor changes in illumination when an image is captured improving model robustness. The weak augmentation is used to not distort pose or depth perception so much that it would contribute negatively to training the model. Validation and test images are not augmented to reflect performance on realistic inputs. We also implement a `MicrorobotDataset` class in order to load images and return tensors for training and evaluation in PyTorch, as well as standardize the depth values. This is done using the mean and standard deviation computed with the training set, which are then also applied to validation, and test set. Data loaders are also created for use in training, validation, and test sets to iterate over the dataset in batches of 32 samples each batch.

IV. Pose Estimation

A. Experiments

Pose estimation is formulated as a multi-class classification problem, with 40 unique poses. Each microscopy image is assigned one pose class with a combination of pitch and roll.

We start by implementing **PoseCNN_V1** (PV1), a baseline CNN consisting of three convolution blocks using `ReLU` after `Conv2d` and followed by `MaxPool2d`. The 64 x 32 x 32 feature map output is then flattened by a fully connected layer that produces 40 logits, one for each pose class.

The second model we implement, **PoseCNN_V2** (PV2), is an extension of the baseline model. We add `BatchNorm2d` after each convolution layer and an additional fully connected hidden `Dropout` layer ($p = 0.2$) before the final classifier layer. This architecture was chosen as a way to regularize and optimize the baseline model by reducing the risk of overfitting. In Figure 4 we are able to see a visual representation of PV2's architecture, which can also help visualize PV1 by excluding `BatchNorm2d` and `Dropout`.

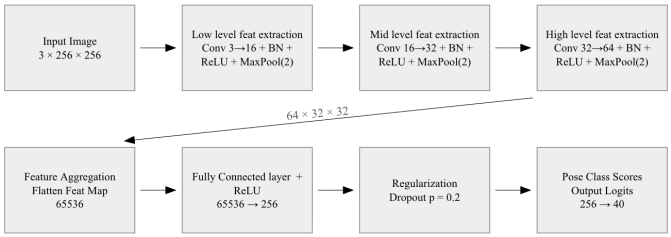

Fig. 4. PoseCNN_V2 Architecture

The last model that we implement and evaluate, **PoseCNN_V3** (PV3), is a standard ResNet18 architecture from torchvision [3] with the final fully connected layer replaced for the output of 40 classes and no pretrained weights.

All models are trained using cross entropy loss and Adam optimizer [4] with a learning rate of 10^{-3}. Accuracy and loss performance are monitored on the validation set at each epoch. We start by visualizing the first five epochs of each model as seen in Table I for PV1. We check if loss decreases and accuracy increases, in order to confirm the model is working properly and we then increase the number of epochs during the actual training.

TABLE I
PoseCNN_V1 Train & Val metrics across epochs

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	3.624	0.092	3.007	0.450
2	1.647	0.568	0.971	0.690
3	0.703	0.752	0.743	0.777
4	0.432	0.845	0.716	0.812
5	0.314	0.907	0.542	0.855

We decide to train each model initially for 15 epochs, prioritizing computational efficiency, printing the values of each epoch as seen in Table I and analyzing whether a plateau is observable. Both PV1 and PV2 were trained for

15 epochs, while because no clear convergence was seen after 15 epochs, PV3 was trained for 30 epochs. PV3 had a much slower learning rate in terms of accuracy increasing and loss decreasing, suggesting that more complex models require more training epochs. For each model, we also record the train/validation loss and accuracy curves to visualize the convergences. As seen in Figure 5, in the last few epochs of the PV1 validation curve for both loss and accuracy does not improve, indicating convergence in performance. A similar pattern is seen in PoseCNN V2 and V3.

In order to optimize and regularize the models, in PV2 we also checked in earlier tests if adding more epochs led to better performance, and although it did increase training accuracy, it led to a steep decrease in validation accuracy suggesting overfitting, therefore choosing to stop at 15 epochs. We also changed the dropout from 0.5 to 0.2 as for previous tests, initially using a dropout of 0.5, the model's training accuracy as well as validation accuracy was increasing at a very slow rate.

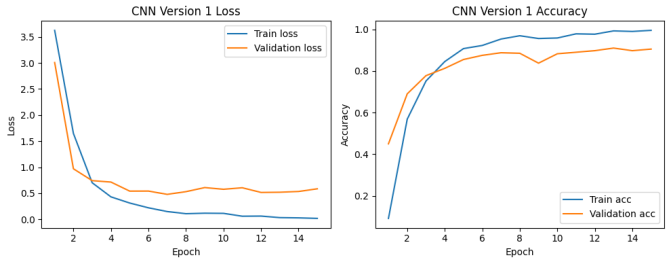

Fig. 5. PoseCNN_V1 Train and Validation Loss/Acc Curve

In Figure 6, we are also able to visualize more fluctuations from epoch to epoch in the validation curves for PV3 model, suggesting instability. This can be due to the complexity of the model, which due to the lack of pretrained weights, and small number of data, did not perform or generalize as well as the simpler CNN versions 1 and 2.

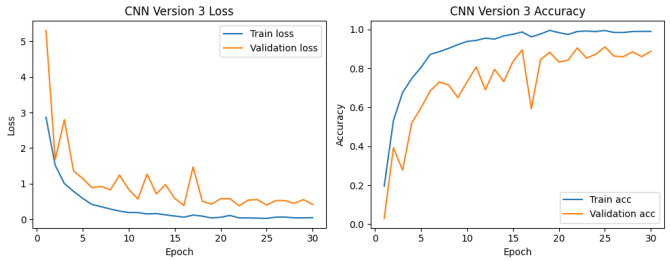

Fig. 6. PoseCNN_V3 Train and Validation Loss/Acc Curve

In Table II we are able to visualize the pose classification performance on the validation set and we are able to observe PV2 achieves the most optimal results.

B. Evaluation

In Table III we note that all PoseCNN versions achieve similar performance on the test sets with PV2 once again showing the highest overall performance, suggesting that the added batch normalization and the hidden layer with the dropout value improved the ability of the model to generalize.

TABLE II
POSE CLASSIFICATION PERFORMANCE ON THE VALIDATION SET

Model	Accuracy	Precision	Recall	F1
CNN V1	0.9050	0.9160	0.9050	0.9037
CNN V2	**0.9050**	**0.9237**	**0.9050**	**0.9054**
CNN V3	0.8875	0.9017	0.8875	0.8860

TABLE III
POSE CLASSIFICATION PERFORMANCE ON THE TEST SET

Model	Accuracy	Precision	Recall	F1
CNN V1	0.8725	0.8892	0.8725	0.8697
CNN V2	**0.8800**	**0.8986**	**0.8800**	**0.8798**
CNN V3	0.8625	0.8919	0.8625	0.8593

In Figure 7, the PV2 confusion matrix is displayed with a very clear diagonal and a few faint off-diagonal entries that can be observed. This reflects the models performance and high accuracy only miss classifying a small percentage of the observations. We can also note that multiple of the off diagonal entries are located near the diagonal suggesting that the model mainly has difficulty with similar pitch and roll configurations that are visually challenging to distinguish.

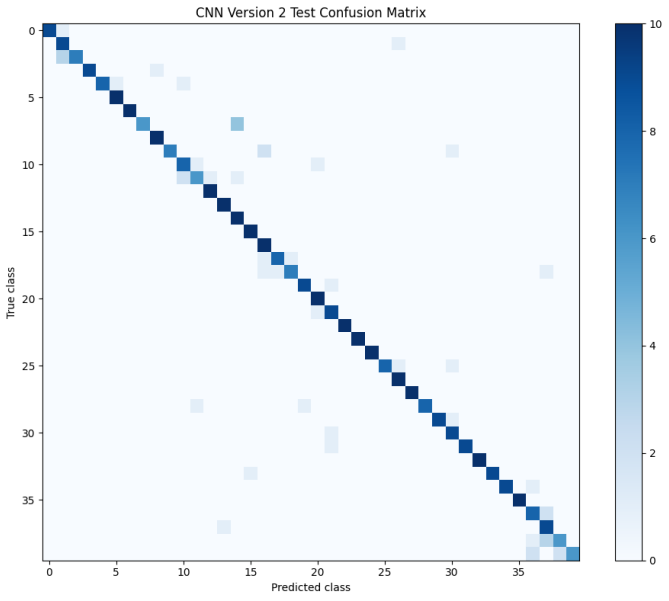

Fig. 7. Confusion Matrix Pose Classification PV2

A limitation to note is that in this project, we formulated pose estimation as a 40 class classification problem, where each class is made of a specific combination of pitch and roll. While doing this did simplify the interpretation of the models, it could have limited performance as the way the data set was naturally set up might have suggested training pitch and roll separately. This can also be seen in the [1] paper, where Wei and Zhang trained pitch and roll separately and

TABLE IV
DEPTHCNN V3 TRAIN & VAL METRICS ACROSS EPOCHS

Epoch	Train MSE	Train R^2	Val MSE	Val R^2
1	0.799	0.201	0.133	0.853
2	0.178	0.822	0.041	0.955
3	0.068	0.932	0.082	0.910
4	0.072	0.928	0.060	0.934
5	0.062	0.938	0.047	0.949

achieved much higher performance metrics close to 100%. Additionally, during training, where all CNN versions reached a higher peak validation accuracy at an epoch that was not the last epoch where the model is saved. For example PV2 reached a higher peak validation accuracy at epoch 14 but for the final evaluation, the epoch 15 version is used. This highlights a limitation that should be fixed using early stopping functions to save the weight from the best validation epoch. Another limitation comes from the nature of microscopic imaging where issues such as defocusing and optical diffraction affect the accuracy of the classification [1].

V. DEPTH ESTIMATION

A. Experiments

Depth estimation is formulated as a regression problem, where we decide to reuse the three models we used for pose classification. The first two models we implement, **DepthCNN_V1** (DV1) and **DepthCNN_V2** (DV2) have the same architecture as PV1 and PV2 respectively. However, instead of the classifier output at the end with 40 logits as output, we have a single output for the standardized depth value of each image. The same is done for **DepthCNN_V3** (DV3) where we use ResNet18 [3]. However, differently from PV3, we decided not to train ResNet18 from scratch, as it performed worst compared to the other 2 simpler CNN models. Therefore, we use pretrained weights to observe whether we are able to output better performance metrics. We also opt for ResNet18 as seen in [1], using deeper architectures benefits depth estimation.

All depth models are trained using mean squared error (MSE) loss and the Adam optimizer [4]. Similarly to pose estimation, we firstly train 5 epochs for each model to check if MSE decreases and R^2 increases which it does in both training and validation for all three models. An example can be seen in Table IV. Recalling one of our limitations from pose classification, we decide to define an early stopping function that returns the best epoch by MSE. Training is allowed to continue for 50 epochs, and early stopping is triggered when the function has not produced a lower MSE than the lowest MSE for more than 5 consecutive epochs. This is applied to all models and we also plot training/validation Loss and R^2 curves as seen in Figure 8 to observe the plateau at which performance metrics converge. We note that DV3, using ResNet18 with ImageNet pretrained weight is able to achieve high R^2 and low MSE much faster compared to DV1 and DV2 supporting the choice of pretrained weights [3].

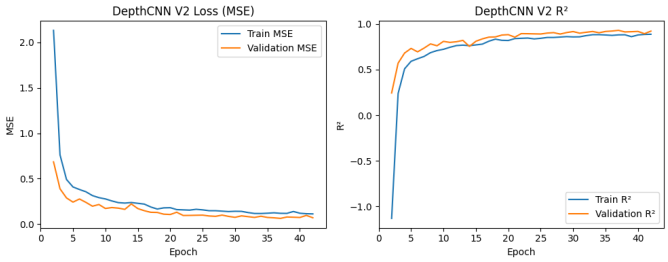

Fig. 8. DepthCNN_V2 Train and Validation Loss/R^2 Curve

TABLE V
DEPTH REGRESSION PERFORMANCE ON THE TEST SET

Model	Test MSE	Test RMSE	Test R^2
DepthCNN V1	0.050	0.223	0.954
DepthCNN V2	0.094	0.306	0.913
DepthCNN V3	**0.028**	**0.169**	**0.974**

B. Evaluation

For each DV, we used the model at the epoch with the best MSE on validation set because of how we defined our early stopping function. In Table V, DV3 performed the best with the lowest RMSE at 0.169 and R^2 of 0.974. This means that it had the lowest error and was able to explain the most variance in depth regression. Although performing relatively well, DV1 still had almost double the error as DV3. In contrast, the weakest performing model was DV2, where the extra regularisation to reduce overfitting might have resulted in slight underfitting, producing higher errors.

An error analysis is computed as seen in Figure 9 for DV3. We observe a true versus predicted depth value graph and a histogram of the error distribution. Both graphs indicate high performance with the true vs predicted graph suggesting that most mistakes occur towards cases with high depth. In the histogram where error is calculated by the difference of the predicted value to the true value, we can observe a bell shaped graph with concentrated values around 0.

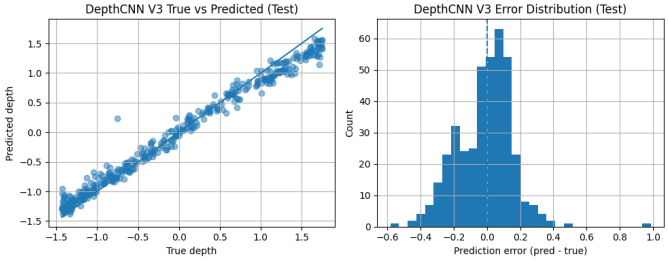

Fig. 9. Ture Vs Predicted & Error Distribution

We observe edge cases in Figure 10. 3 of the 5 highest edge cases have very similar poses, which suggests an inaccuracy due to the pose making it hard to determine the true depth value. Two of the 3 observations have the same true depth, suggesting that at that depth value the model is not able to properly analyze the image or recognize that pattern.

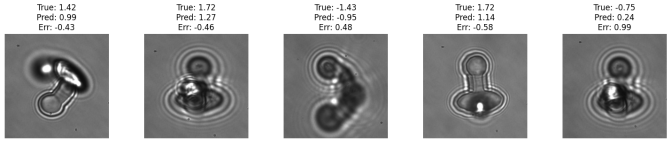

Fig. 10. Top 5 Edge Case Images

TABLE VI
HYPERPARAMETER TUNING RESULTS FOR DEPTHCNN V3

lr	weight decay	Epoch	Best Val MSE	Best Val R^2
0.0010	0.0000	3	0.032	0.964
0.0010	0.0001	4	0.029	0.968
0.0001	0.0000	15	**0.015**	**0.984**
0.0001	0.0001	10	**0.015**	**0.984**

Seeing that the edge cases have a similar pose, we observe depth distribution for each pose in Figure 11. In multiple of the edge cases the true depth value for that specific pose had no training observations close to that value visualized by the gaps in those distributions. For example, we take a look at the train depth distribution for pose where the true value is 1.720, but as seen in the histogram for pose 17 the standardized depth does not reach a value over 1.5. Instead, looking at the predicted value, 1.138, the number of observations for that value or close to that value is very high. A similar observation can be made for all other edge cases except for pose 26.

A limitation to consider is that this superior performance of DV3 compared to previous DV1 and DV2 is due not only to the added complexity in the architecture of ResNet18 [3] but also to the extra information from the pretraining that is not available to the simpler CNN models.

VI. HYPERPARAMETER TUNING

For hyperparameter tuning, we use our best model which was the pretrained ResNet18 for depth estimation or DV3 and the parameters we tune are the learning rate as well as L2 weight decay to see if they can contribute to a better performance. We observe the results in Table VI.

From the results we visualize the best MSE and R^2 at each parameter with varying learning rate between 10^{-3} which was what we used for all the previous models vs 10^{-4} which we have not used before, as well as adding weight decay for each learning rate and comparing it with no weight decay. Table VI illustrates that with a lower learning rate the validation results are substantially better compared to the higher learning rate of 10^-3, having around half the error at a MSE of 0.015. We note that weight decay has little impact on both learning rate, with the 10^{-3} learning rate obtaining slightly better performance with weight decay, and the 10^{-4} learning rate achieving the same performance with weight decay but requiring fewer epochs to reach that performance. This minimal effect could be due to the small degree at which weight decay is applied. However, weight decay being able to regularize large weights still allows the model to reach a similar optimal result with fewer epochs. The pretrained ResNet18 performing better with a lower learning rate suggests that with the higher learning rate, the model is not able to properly fine tune the existing features and instead partially overcorrect and catch unnecessary noise that a smaller learning rate is able to be flexible about. We proceed to using lr = 10^{-4} and wd = 10^{-4} to evaluate the results on the test set as it achieved the best validation results with the highest computational efficiency.

TABLE VII
TUNED DEPTHCNN_V3 HYPERPARAMETERS AND TEST PERFORMANCE

lr	weight decay	Epoch	MSE	RMSE	R^2
0.0001	0.0001	10	0.019	0.137	0.983

As seen in Table VII, evaluating the results from the test set using the tuned DV3, we achieve a better result with an RMSE of 0.137, compared to the original untuned DV3 with an RMSE of 0.169.

VII. CONCLUSION

In conclusion, this report focused mainly on two tasks: **Pose Estimation**, a 40 class classification problem, and **Depth Estimation**, a regression problem. For both tasks, we compared two custom CNN baseline models and a ResNet18 [3] model. In pose estimation, the regularized CNN PV2 achieved the strongest overall performance, with a test accuracy of 0.88 and a per class average F1 of 0.8798, suggesting the added BatchNorm and dropout imporved the generalization ability compared PV1. In contrast, PV3 without pretrained weights performed poorly, suggesting that deeper architectures may require more data or stronger augmentation.

On the other hand, depth estimation displayed a considerably better performance, with DV3 or ResNet18 based model achieving the lowest error among the three models evaluated with a test RMSE of 0.169 and R^2 of 0.974. The results are consistent with the findings of Wei and Zhang in their paper [1] that defines deeper architecture models as advantageous for depth regression. Using hyperparameter tuning, we were also able to improve DV3 performance reducing RMSE to 0.137 and increasing R^2 to 0.983, highlighting the importance of using parameters to optimize model performance.

VIII. FUTURE WORK

Several points of improvements are observed during the evaluation sections of our project. The performance metrics in pose classification are rather disappointing considering the results achieved in the [1] paper. This could have been due to training pose classification as a single 40 class label instead of using pitch and roll separately as seen in [1]. Furthermore, early stopping could also be applied to the pose classification and not just depth estimation. Data augmentation could also be executed to a greater extent to see whether it could improve performance metrics. Due to computational efficiency problems, we were also not able to perform hyperparameter tuning to a deeper degree, with changing batch sizes or even using different models such as ResNet50 or EfficientNet which

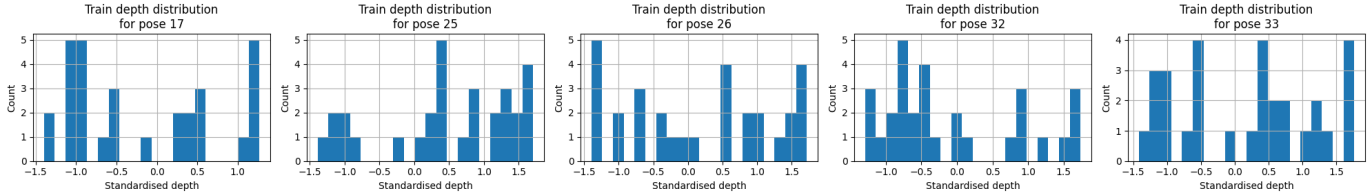

Fig. 11. Edge Case Pose Distribution

could have optimize performance. One additional point to note is that in future projects, k-fold cross validation can be considered instead of using one fixed validation set. This could provide more reliable estimates of how a model generalizes instead of relying on only one single validation set. However, computational efficiency to performance tradeoff should be considered.

REFERENCES

[1] L. Wei and D. Zhang, "A dataset and benchmarks for deep learning-based optical microrobot pose and depth perception." arXiv:2505.18303, 2025. Accessed: 2025-12-12.

[2] scikit-learn developers, "Common pitfalls and recommended practices." scikit-learn documentation, 2025. Accessed: 2025-12-12.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.