3D Object Classification and Segmentation

Ceren Doner
MSc Robotics and AI, UCL CS
ceren.doner.24@ucl.ac.uk

Lorenzo Uttini
MSc Robotics and AI, UCL CS
lorenzo.uttini.24@ucl.ac.uk

Amlan Sahoo
Msc Robotics and AI, UCL CS
amlan.sahoo.24@ucl.ac.uk

I. Introduction

Recent advances in deep convolutional networks have significantly improved performance in 2D vision tasks like object detection and segmentation. However, for applications that require spatial reasoning—such as robotics, autonomous navigation, and indoor mapping—2D methods alone are often insufficient. These domains benefit greatly from 3D scene understanding, where depth cues and geometric structure play a critical role.

Point clouds, obtained from RGB-D sensors or LiDAR, offer a compact and informative representation of 3D environments by capturing the spatial layout of surfaces. In this report, we focus on the binary classification and segmentation of indoor scenes to detect the presence and extent of **tables** using 3D point cloud data. We utilize nine annotated sequences from the Sun3D dataset [1]: five from MIT for training and four from Harvard for testing, including sequences without tables to serve as negative examples. These datasets represent diverse indoor environments containing structured geometric objects (e.g., desks, dining tables), making them suitable for learning-based 3D scene understanding.

While traditionally geometry-based methods [2] were used for object detection in point clouds, the emergence of *PointNet* [3] pioneered point-based deep learning methods for 3D interpretation. However, it lacks mechanisms to capture local geometric context, which is crucial for detecting structured objects like tables. Therefore, we adopt *DGCNN* [5], a dynamic graph convolutional network that aggregates edge features to capture local surface information, making it particularly suitable for segmentation and classification of structured objects with sharp boundaries.

This report explores three distinct pipelines:

- **Pipeline A**: Converts depth to point cloud for binary scene classification
- **Pipeline B**: Predicts depth from RGB using monocular depth estimation, then performs scene classification
- **Pipeline C**: Converts depth to point cloud and applies point-wise segmentation

Each pipeline enables exploration of different 2D-to-3D reasoning strategies for scene understanding, contributing insights into the strengths and limitations of 3D deep learning for indoor environments.

II. Dataset Processing

The Sun3D annotated dataset comprises indoor scenes captured across various buildings. While the original dataset includes 14 semantic class labels, our focus is solely on a generalized "table" class, treating all other categories as background. This effectively reduces our task to binary classification/segmentation. The dataset contains 281 training samples located under the *mit* folders and 98 test samples under the *harvard* folders.

Seven images were excluded from the training set due to missing annotations for the table class. A quick visual inspection revealed some notable similarities between the indoor environments across the two domains, particularly in scenes featuring tables. However, it remained generally evident that the images originated from different buildings.

Additionally, we also collected 50 RGB-D samples at University College London (UCL) using a D455 RealSense setup. These samples were used as supplementary test data to qualitatively evaluate our model's generalization in real world conditions.

A. Preprocessing for Binary Classification from Depth Images

Firstly, we applied data augmentation to the depth images. This is a standard technique in image-based tasks, as it helps improve the model's robustness by introducing variability in the training data.

The aim of Pipeline A is to perform binary classification by taking depth images as input, converting them into point clouds, and using a classification model to predict whether the image contains a table ('1') or not ('0'). This pipeline focuses on the geometric structure of the scene provided by the depth channel independent of RGB information.

To make this transformation, camera intrinsics are loaded from intrinsics.txt file for each sequence, which are essential for converting 2D pixel coordinates in the depth image into real-world 3D coordinates. The transformation is carried out using the pinhole camera model, where each pixel with a valid depth value is mapped into a 3D point (X,Y,Z) generating a dense point cloud representation of the scene.

After generating the full point cloud, a uniform downsampling is applied to reduce the point cloud to a fixed number of points (1024 in our case). This is necessary to have a consistent input shape for the classification network and to reduce computational load. Without this step, variations in point cloud size across images would make training unstable or computationally expensive.

The annotation file associated with each sequence contains polygon data defining the 2D contours of table surfaces within the image. Using this information, we filter the 3D point cloud

and retain only those points whose original pixel locations fall inside these polygon boundaries. In the case of images labeled as containing a table (label = 1), we extract only the point cloud segment that falls within the annotated polygon and isolate the table-shaped region. Conversely, for images without any table annotation (label = 0), we use the entire background point cloud as it is. This step is important; otherwise, the model would be trained on large, unstructured point clouds without consistent shape which makes it significantly harder to learn meaningful representations. By focusing on the geometrically consistent region for positive samples and background context for negative ones, the model learns to differentiate table instances from non-table scenes more effectively.

The dataset had challenges due to its small size and some missing or incorrect annotations, which were filtered out to avoid misleading the model.

B. Preprocessing for Monocular Depth Estimation from RGB

This set of preprocessing steps is applied in the first stage of *PipelineB*, where the goal is to estimate depth maps from input RGB images using a monocular depth estimation model.

After generating the predicted depth maps, we applied outlier removal using percentile clipping (3rd to 97th percentile). This helped reduce noise and improve the quality of the depth maps before they were passed to the classification stage.

To enable a fair comparison between predicted and ground truth depth maps during evaluation, we converted the ground truth depths, originally stored in decimillimeters for the SUN dataset and in centimeters for the RealSense dataset, to meters—matching the scale of the predicted outputs. Additionally, since the predicted depth maps had a resolution of $(384, 512)$ pixels, we resized them to match the resolution of the ground truth images, which was $(480, 640)$.

Finally, we applied a validity mask to both predicted and ground truth maps to exclude invalid pixels from evaluation. This was necessary because, in many cases, invalid pixels accounted for at least 10% of the ground truth depth images.

C. Preprocessing for Segmentation (from Depth Images)

Similar to Pipeline A, we first apply random augmentations to the training data to increase variablity. But unlike Pipeline A, we do not incorporate image level class balancing. That is because segmentation is focused on pixel level distribution so the imabalance at the pixel level is more important to us. Therefore, we implement direct proportional weighting which allows the model to focus on regions more likely to belong to the table, amplifying confident predictions and improving spatial precision. Subsequently, the augmented 2D depth maps are transformed into a set of 3D points corresponding to each pixel. The point clouds are then labeled based on polygon annotations provided in the corresponding datasets as described above.

Also in contrast to uniform downsampling, here we employ a class balanced downsampling that first selects a subset of points and then ensures that the ratio of table versus background points remains balanced. This was critical because we essentially wanted our model to learn "table" features and the point cloud was dominantly containing background points. Therefore we set a table to non-table ratio of 0.7 to ensure tables are more represented in the downsampled point cloud and thus mitigate class imbalance issues on pixel levels during training.

III. Methods

A. PipelineA

For the first pipeline, our input consists of 3D point clouds derived from depth images. Each point cloud is labeled as 1 if the scene contains a table, and 0 if there is no table present. This binary labeling allows the model to learn patterns in the shape and structure of tables compared to generic background regions.

Our training strategy involves using 5-fold Stratified Cross-Validation. This method is ideal for classification tasks when working with imbalanced datasets as it ensures each fold has a representative ratio of both classes [10]. This approach also helps improve the generalization performance of the model and provides a more reliable estimate. In each fold, the dataset is split into 220 samples for training and 55 for validation, and we select the best-performing model checkpoint based on highest validation accuracy with lowest validation loss.

The DGCNN model was trained using the following hyperparameters: a batch size of 16, a learning rate of 0.0001 and 50 training epochs. We used CosineAnnealingLR as the learning rate scheduler to allow smooth learning rate decay over time [11]. To reduce the risk of overfitting, a dropout rate of 0.5 was applied in the fully connected layers and L2 regularization was used with the weight decay of 1×10^{-4}. The optimizer used was Adam, which provides fast convergence, and the loss function was weighted CrossEntropyLoss, which takes class imbalance into account.

Since our training set consists of only 275 samples, we computed the class distribution and applied class weighting. The dataset contains 115 samples for class 0 (no table) and 160 samples for class 1 (table), corresponding to 41.82% and 58.18%, respectively. To balance this distribution during training, we applied inverse frequency weighting to the loss function:

- Class 0 weight: 1.1957
- Class 1 weight: 0.8594

These weights were passed into the CrossEntropyLoss to make sure that the model does not become biased toward the more frequent class allowing it to learn a fair decision boundary between the two classes.

B. PipelineB

The goal of PipelineB is to classify whether a given RGB image contains a table. To approach this task efficiently, the pipeline is divided into three main stages:

1) **From RGB to Depth Maps**: Starting from a single RGB image (monocular), a depth map is generated using a pretrained depth estimation model.

2) **From Depth Maps to Point Clouds**: The generated depth maps are converted into point clouds using the dataset's camera intrinsics and geometric transformations.
3) **From Point Clouds to Binary Classification**: Finally, the point clouds are fed into our pretrained classification model from Pipeline A to predict a binary label: table or no table.

Since the second and third stages have already been described in Section III-A, this section will focus on providing a detailed explanation of the first stage, including the methodology and implementation choices.

Our dataset consists mainly of indoor scenes showing rooms that typically include a table. We do not apply any initial transformations to the input RGB images; they are fed directly into the pretrained model used for depth estimation.

For this task, we chose to use ZoeDepth [9], a zero-shot transfer model capable of estimating depth from a single RGB image. ZoeDepth combines both relative and metric depth estimation, achieving strong generalization performance while preserving metric scale accuracy.

It consists of an encoder-decoder architecture enhanced with MiDaS blocks for extracting relative depth. A separate metric bins module generates a per-pixel depth distribution, which is combined with the relative depth map via a softmax-weighted sum to produce metric depth. Residual connections and multi-scale features improve spatial accuracy. The final output is a dense metric depth map aligned with the input RGB image.

We chose this model due to its ability to output absolute depth map estimations, which allows us to directly compare the predictions with ground truth depth maps—an essential requirement in our case.

After predicting the depth maps, we needed to match these estimations with the ground truth depth images already provided by the dataset. First, as mentioned in Section II-B, we removed outliers by clipping the 3rd and 97th percentiles. This step was necessary to mitigate the effect of noise introduced during prediction and to ensure that the most meaningful depth values were preserved.

Next, we applied a validity mask to both the ground truth and the predicted depth maps. We observed that ground truth images contained a significant number of invalid pixels (approximately 10% per image). It was therefore crucial to define a suitable masking strategy for each dataset to exclude these invalid pixels. After several trials, we selected the following valid depth ranges:

- *SUN3D*: gt_depth > 0.6 & pred_depth > 0.01
- *RealSense*: $0.04 <$ gt_depth < 0.30 & pred_depth > 0.01

To align the predicted depth maps with the ground truth in metric scale, we computed a scale factor by dividing the median of the ground truth depth values by the median of the predicted depth values over the valid pixels.

After computing different evaluation metrics on our predicted depth maps, we can pass to the second and third stages of the pipeline. From this point onward, we employed the same model trained in Section III-A and we run inference on our predicted depth maps. This is followed by a quantitative evaluation of the binary predictions and comparison with the results of Pipeline A.

C. PipelineC

The segmentation model is based on the Dynamic Graph Convolutional Neural Network (DGCNN) architecture, customized to segment between table and background classes:

Network Architecture: The model constructs a dynamic graph on the input points by computing k-nearest neighbors. Each convolutional layer (Edge Convolution Block) captures local geometric relationships by applying convolution over the edges of the graph. The network stacks multiple such blocks to aggregate features from varying scales and then consolidates these features using point-wise multi-layer perceptrons (MLPs) for segmentation.

Loss Function and Class Imbalance: The network outputs per-point predictions. During training, these predictions are reshaped and compared against the ground truth labels using a weighted cross-entropy loss. The weight for the table class is doubled to address the typical point level class imbalance observed between table and non-table points in the dataset.

Training Procedure: The hyperparameter selections and training procedure is nearly identical to that of Pipeline A. Both the training and validation losses, along with accuracy and IoU metric for the table class, are logged. Checkpoints are saved when the model achieves a lower validation loss or a higher IoU.

Evaluation and Visualization: The model is then assessed on the Harvard test sequences using an evaluation script. The pipeline computes metrics including point level prediction accuracy and class-specific IoUs for tables and background. Visualization utilities further allow inspection of the segmentation results by comparing predictions on the original depth images and visualizing the predicted 3D point cloud with Open3D.

IV. Results

A. Pipeline A and B: Classification

1) Quantitative Analysis: After training on the MIT dataset in Figure 1, we evaluated the model using 5-fold Stratified Cross-Validation. Fold 4 achieved the best validation accuracy with low validation loss, making it the best choice for testing. The loss and accuracy curves showed smooth convergence without signs of overfitting or underfitting. Overall, the model performs consistently across different data splits, even with limited training samples.

Metric	Pipeline A	Pipeline B
Accuracy	0.97	0.93
Precision	1.00	0.94
Recall	0.96	0.93
F1	0.98	0.93

TABLE I: Comparison between pipelineA and pipelineB on evaluation classification metrics on Harvard test datasets

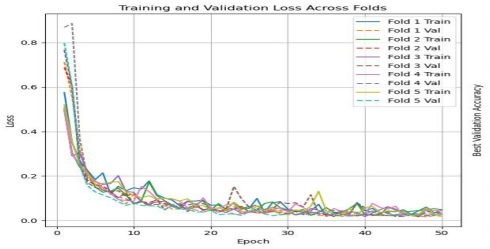

Fig. 1: Training and Validation Loss Across Folds for PipelineA

As can be seen from the Table I, Pipeline A performs better than Pipeline B across all evaluation metrics. This suggests that using real depth images as input, as done in Pipeline A, results in more accurate point cloud generation and subsequently better classification performance. Pipeline B, which relies on estimated depth maps from RGB images, although not far behind, might still suffer from lower-quality depth representations that can affect the model's ability to distinguish table structures especially in complex scenes.

2) Qualitative Analysis: We visualized both correct and incorrect predictions made by the model on the Harvard test set and our custom RealSense dataset.

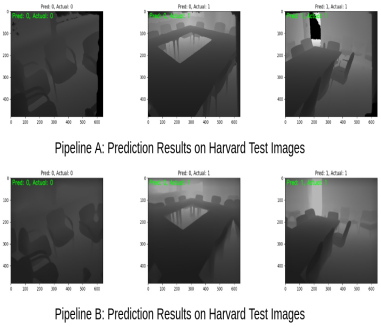

Fig. 2: Comparing Results on Harvard Test Images

From these examples in Figure 2, we can see that both pipelines produce similar results in distinguishing scenes without tables, especially in cases where only chairs are present. Since similar chair-only scenes are included in the training data, the model appears to have learned to recognize this pattern as a non-table case. In contrast, scenes containing tables are correctly classified, most likely because the model has learned to identify smooth surfaces and rectangular shapes with defined corners which are common features of tables in the training set.

Another common misclassification observed was with the case of concentric recatangles. In such cases, both pipelines tend to predict "no table" because the model mainly learned from continuous rectangular tables with simple shapes during training. As a result, it struggled to generalize to configurations that differ from those seen in the training set.

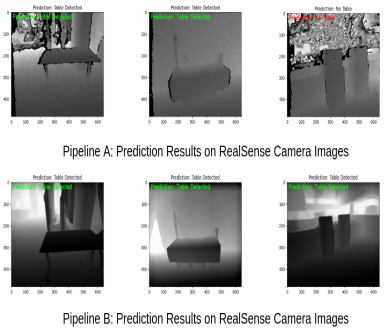

Fig. 3: Comparing Results on RealSense Camera Images

In the RealSense dataset (Figure 3), the model captured the patterns it was trained on primarily recognizing simple rectangular-shaped objects with smooth surfaces, similar to those seen during training. It also responded positively to structurally simple shapes and in some cases mistakenly predicted rectangular boxes as tables due to their smooth and flat surfaces. This indicates that the model tends to classify detected objects with simple geometric characteristics as tables, even when they are not. However, the RealSense dataset is noisier and more challenging, as the depth images are generated directly from raw scenes making point cloud prediction less accurate compared to pre-processed harvard datasets.

Interestingly, in one case where a TV appeared in the scene, Pipeline A correctly predicted "no table." Although the TV had a similar rectangular shape, background noise in the depth image likely disrupted the shape information, causing the model to treat it as part of the background. This shows Pipeline A's ability to reject misleading inputs due to its reliance on real depth data, even when visual shapes are similar to tables. In contrast, Pipeline B classified the same scene as containing a table. Since it uses RGB-based depth estimation, it is more sensitive to noise and less accurate depth representation. This often leads the model to interpret any smooth, rectangular object as a table even when it's not. This comparison shows how input quality and noise sensitivity can affect classification behavior between models trained on true depth versus generated depth.

B. Pipeline B: Depth Estimation

To evaluate the performance of the pretrained model on our RGB images, we compared the predicted depth maps with the ground truth depth maps provided by the datasets (SUN3D and RealSense). The evaluation was carried out using 4 commonly used metrics in monocular depth estimation ([6], [7]):

Absolute Relative Error (AbsRel): Average of the absolute difference between predicted and ground truth depth, divided by the ground truth. Lower values indicate better performance.

Root Mean Squared Error (RMSE): Square root of the average squared differences between predicted and ground truth depth values, penalizing larger errors more heavily.

Mean Absolute Error (MAE): Average of the absolute differences between predicted and ground truth depths.
log10 Error: Average error between the $\log_{10}$ values of predicted and ground truth depths, highlighting relative scale inconsistencies.

Table II presents the results obtained across the SUN dataset, split between the training set (*MIT*) and test set (*Harvard*), as well as the *RealSense* dataset.

Metric	Harvard	MIT	RealSense
AbsRel	0.1903	0.2380	0.1524
RMSE	0.5563	0.4387	0.3587
MAE	0.3340	0.2581	0.2330
Log10	0.0690	0.0508	0.0802

TABLE II: Averaged depth evaluation metrics across different datasets. All values are in meters.

As shown by the *AbsRel* metric, the model performs well across all datasets, with relative errors below 20% for both *Harvard* and *MIT*, and below 16% for *RealSense*. A similar trend is observed for *RMSE* and *MAE*, where *RealSense* shows the lowest errors—approximately 36 cm root mean squared error and 23 cm mean absolute error. In contrast, *MIT* and *Harvard* exhibit slightly higher values, with RMSEs around 0.4–0.5 and MAEs around 0.26–0.35 meters.
The final metric, *Log10*, follows a different pattern, with a higher score for *RealSense*, 0.08, and better scores for MIT and Harvard with around 0.07 and 0.06 for Harvard and MIT, respectively.
Overall, the pretrained model demonstrates strong generalization and performance across all datasets, especially on *RealSense*, where the predicted maps are notably accurate. These findings are further supported by visual inspection. Figure 4 displays a side-by-side comparison between a predicted depth map and its corresponding ground truth.

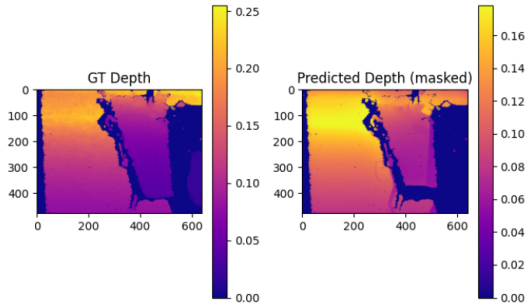

Fig. 4: Depth comparison between GT and Predicted Map

We can observe that the predicted image closely resembles the ground truth. Aside from a slightly lower scale and subtle variations in some areas, the overall structure and depth consistency of the prediction make it a reliable approximation of the real depth.

C. Pipeline C: Segmentation

The following observations were made during training the segmentation network:

Loss: Both training and validation loss steadily decreased over 50 epochs (from 0.38 to 0.10 and 0.89 to 0.10, respectively). Interestingly, the validation loss was initially lower than the training loss in early epochs for most runs, a common occurrence in small datasets due to "easy" validation examples or dropout regularization which was applied only during training. Despite this, convergence was stable, and both losses became nearly equal by epoch 50.
Accuracy: Training and validation accuracy remained consistently above 90%, indicating correct classification of most points. However, as accuracy can be misleading in imbalanced segmentation tasks, we focus on IoU as the main metric.
IoU: The Intersection-over-Union (IoU) for the *table* class peaked at 0.91 (epoch 47) and ended at 0.90 by epoch 50. We selected the model from epoch 47 as our best checkpoint.

All training metrics log and plots are available under *src/pipelineC/checkpoints*.
We then evaluated the best model on the 4 Harvard test sequences. Table III summarizes the results:

Sequence	Accuracy	Table IoU	Background IoU
harvard_c5	91.62%	0.8785	0.7906
harvard_c6	92.02%	0.8846	0.7944
harvard_c11	97.22%	0.9576	0.9254
harvard_tea_2	94.51%	0.00	0.9451

TABLE III: Test metrics on Harvard sequences: Accuracy, Table IoU and Background IoU
The segmentation network achieves an overall average accuracy of 92% and Table IoU of 87.2%, demonstrating strong generalization to unseen data. Figure 5 shows an example segmentation result with excellent accuracy. The entire predictions are available in *pipelineC/results/visualizations*.

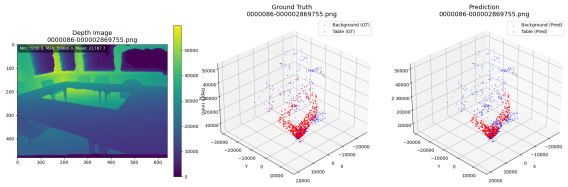

Fig. 5: Sample Segmentation Prediction on Harvard data

We also conducted a qualitative evaluation on a real-world RGBD capture from a RealSense camera, shown in Figure 6: This shows that the segmentation fails significantly; nearly all points are classified as table, regardless of context. This suggests the model possibly struggles to generalize beyond the distribution of the Sun3D dataset.

V. Discussion

A. Classification Results for Pipeline A and Pipeline B

In the previous section, we presented the results of our classification models for both Pipeline A and Pipeline B. As described earlier, while A directly uses ground truth depth images to generate point clouds that are then fed into a

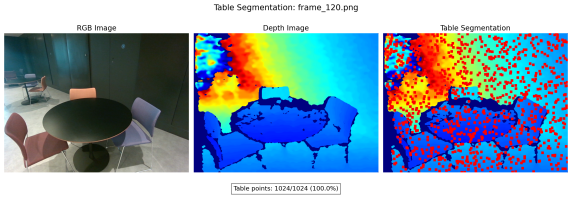

Fig. 6: Pipeline C inference on a UCL sequence

DGCNN classifier. In contrast, B first estimates depth maps from RGB images using a pretrained model (ZoeDepth) and then applies the same classification as in Pipeline A.

Table I shows that both pipelines achieved excellent performance on the Harvard dataset, with near-perfect scores on evaluation metrics such as Accuracy, Precision, Recall, and F1-score. These results indicate that the model trained in Pipeline A generalizes extremely well to unseen data, while the high performance of Pipeline B confirms that the ZoeDepth predictions are very similar to the GT depth maps.

Notable differences arise on the RealSense dataset. As illustrated in Figure 3, the GT depth maps acquired with the RealSense camera are significantly noisier than the predicted ones. Despite the smoother appearance of the ZoeDepth outputs, Pipeline B misclassifies several images as containing tables, resulting in an increased number of false positives. In contrast, Pipeline A, which uses the noisier ground truth maps, correctly labels some images as negative, likely because the noise obscures spurious details that could be misinterpreted as table contours. These observations suggest that the differences in scene composition, sensor characteristics, and data acquisition (e.g., camera calibration and lighting conditions) between the datasets have a significant impact on the classification performance of each pipeline.

These results also highlight the trade-off between accuracy and flexibility when comparing the two pipelines. Pipeline A benefits from using real depth data, which provides more accurate 3D information. This helps the model better understand the shape of objects in the scene, especially in complex environments like those found in the RealSense dataset. As a result, it makes fewer mistakes and achieves higher precision. However, Pipeline A depends on having a depth sensor, which can limit its use in situations where only RGB images are available. In contrast, Pipeline B is more flexible because it can work with just RGB images by estimating the depth. This makes it useful in real-world cases where adding special hardware is not possible. Still, the predicted depth maps are less precise and can be affected by lighting, scene clutter, or textures in the image. This can lead to more incorrect predictions, as seen in the lower precision and F1 scores. Overall, the choice between the two pipelines depends on the use case: Pipeline A is better when depth data is available and accuracy is critical, while Pipeline B is a good option when working with RGB images only and more general use is needed.

B. Pipeline C

The segmentation network performs well on the Harvard sequences, particularly in scenes structurally similar to the training data. The highest IoU is achieved in `harvard_c11`, likely due to its spatial similarity to the MIT training sequences. Conversely, `harvard_tea_2` has no table objects—resulting in a 0.00 Table IoU—and also shows false positives (6% of points incorrectly labeled as table). This highlights the challenge of evaluating performance on sequences with no positive class examples.

Several factors would have possibly contributed to the success on the standard dataset, including the the use of edge convolutions allowing the model to capture local surface geometry, which is essential for segmenting flat, structured objects like tables. The loss function weighted the table class more heavily (weight = 2.0), helping mitigate class imbalance, while random transformations (rotation, jitter, scaling) improved the model's robustness to orientation and size variations. Additionally as the dataset was small, regularization (dropout) and weight decay prevented overfitting explaining the exceptional test results.

However, generalization to custom RealSense data was poor. Likely causes include:

Domain Gap: The Sun3D and RealSense data differ significantly in terms of sensor characteristics, lighting, and scene layout.

Noisy Depth Maps: RealSense data includes depth noise, invalid points, and sparsity which might have led to unreliable point cloud representations.

Limited Diversity: The training data may not have adequately capture the variability in real-world scenes.

VI. Conclusions

In this work, we explored three pipelines for detecting tables in indoor 3D scenes: Pipeline A used ground truth depth to classify images, Pipeline B used RGB images with predicted depth, and Pipeline C performed point-level segmentation. Pipeline A gave the best accuracy when high-quality depth was available, while Pipeline B offered more flexibility using only RGB images. Pipeline C performed well on segmentation when tested on standard data.

However, there were some limitations. The classification models struggled to generalize to noisy real-world data from RealSense sensors. Pipeline C, in particular, overfitted to the SUN3D training set and did not perform well on new scenes. Also, all pipelines relied heavily on accurate camera intrinsics and preprocessing steps, which can be error-prone.

Future improvements could include domain adaptation to reduce the gap between clean and noisy data, and fine-tuning ZoeDepth on a few labeled RealSense samples. Using self-supervised learning or synthetic data could also help the models perform better when data is limited or noisy, and make them more general across different environments.

References

[1] J. Xiao, A. Owens, and A. Torralba, "SUN3D: A Database of Big Spaces Reconstructed Using SfM and Object Labels," in Proc. ICCV, 2013.

[2] A. Nguyen and B. Le, "3D point cloud segmentation: A survey," 2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM), Nov. 2013.
[3] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep Learning for 3D Point Clouds: A Survey," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020.
[4] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," arXiv preprint arXiv:1612.00593, 2016.
[5] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic Graph CNN for Learning on Point Clouds," arXiv:1801.07829 [cs], Jun. 2019.
[6] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 27, 2014.
[7] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *Proc. of the International Conference on 3D Vision (3DV)*, 2016, pp. 239–248.
[8] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.
[9] S. F. Bhat, R. Birkl, D. Wofk, P. Wonka, and M. Müller, "ZoeDepth: Zero-shot transfer by combining relative and metric depth," *arXiv preprint arXiv:2302.12288*, 2023.
[10] S. Szeghalmy and A. Fazekas, "A Comparative Study of the Use of Stratified Cross-Validation and Distribution-Balanced Stratified Cross-Validation in Imbalanced Learning," Sensors, vol. 23, no. 4, p. 2333, Feb. 2023.
[11] Ohamouddoua, Said, et al. "Introducing the Short-Time Fourier Kolmogorov Arnold Network: A Dynamic Graph CNN Approach for Tree Species Classification in 3D Point Clouds." arXiv preprint arXiv:2503.23647 (2025).