COMP0248 Group D: 3D Point Cloud Table Detection and Segmentation

This repository contains implementations of three complementary pipelines for detecting and segmenting tables in indoor scenes:

1. **Pipeline A**: Point Cloud Classification with DGCNN (determines if a scene contains a table)
2. **Pipeline B**: Depth Estimation and Point Cloud Classification (estimates depth from RGB images)
3. **Pipeline C**: Point Cloud Segmentation with DGCNN (segments table vs. background points)

Directory Structure

```
project-root/
├── data/
│   ├── CW2-Dataset/           # Contains Sun3D sequences (deleted for
submission) and predicted depth images from Pipeline B
│   │   ├── mit_*/             # MIT training sequences with
depth/RGB/annotations
│   │   └── harvard_*/         # Harvard test sequences with
depth/RGB/annotations
│   └── RealSense/             # Custom UCL sequences collected on
RealSense D455 RGB-D
│       └── UCL_Data/          # Custom dataset with RGB and depth folders
├── src/
│   ├── pipelineA/             # Table classification
│   │   ├── dataloader.py                   # Preprocessing for
MIT/Harvard datasets
│   │   ├── dataloader_custom.py            # Preprocessor for RealSense
data
│   │   ├── model.py                        # DGCNN classifier
implementation
│   │   ├── train.py                        # Training with 5-fold cross-
validation
│   │   ├── test.py                         # Evaluator for MIT/Harvard
test data
│   │   └── test_custom.py                  # Inference on RealSense data
│   ├── pipelineB/             # Depth estimation + Table Classification
│   │   ├── dataloader.py                   # Data preprocessing for
MIT/Harvard
│   │   ├── dataloader_custom.py            # Data loader for RealSense
dataset
│   │   ├── model.py                        # DGCNN classifier
implementation
│   │   ├── depth.py                        # Depth estimation for
MIT/Harvard
│   │   ├── depth_custom.py                 # Depth estimation for
RealSense
│   │   ├── test.py                         # Testing on MIT/Harvard
datasets
```

```
│   │   ├── test_custom.py                        # Testing on RealSense dataset
│   │   └── utils/                                # Helper modules for depth
processing
│   │       ├── depths.py                         # Depth loading and processing
│   │       ├── metrics.py                        # Metrics for depth evaluation
│   │       └── visualization.py                  # Depth visualization tools
│   └── pipelineC/             # Table Segmentation
│       ├── checkpoints/                          # Saved model weights and
training curves
│       ├── results/                              # Evaluation results &
visualizations
│       ├── utils/                                # Helper modules
│       │   ├── metrics.py                        # Segmentation evaluation
metrics
│       │   └── visualization.py                  # Point cloud & segmentation
visualization
│       ├── models/                               # Model architecture
│       │   └── dgcnn.py                          # DGCNN segmentation
implementation
│       ├── dataset.py                            # Point cloud segmentation
dataset
│       ├── evaluate.py                           # Evaluator for Harvard test
data
│       ├── inference.py                          # Inference on RealSense
custom data
│       └── train.py                              # Training with
train/validation split
├── README.md
└── requirements.txt                              # Project dependencies
```

Requirements

- Python 3.6+
- PyTorch
- NumPy
- OpenCV
- Matplotlib
- Scikit-learn
- Seaborn
- tqdm
- Transformers
- Open3D

```
pip install torch numpy opencv-python matplotlib scikit-learn seaborn tqdm
transformers open3d
```

Pipeline A – Point Cloud Classification Using DGCNN

This pipeline determines if a scene contains a table by converting depth images to 3D point clouds and classifying them with a Dynamic Graph CNN model.

Features

- **Data Preprocessing**: Converts depth images to 3D point clouds with augmentation
- **5-fold Cross-validation**: Ensures robust model evaluation
- **Visualization**: Generates comprehensive performance visualizations

How to Run

1. Data Preprocessing

The data preprocessing is handled in the `dataloader.py` file. This file defines functions to:

- **Load intrinsics:** Reads camera intrinsics (or uses default values).
- **Convert depth images to 3D point clouds:** The `depth_to_pointcloud` function computes 3D coordinates from a given depth image.
- **Downsample & Augment:** The `downsample_pointcloud` and `random_augmentation` functions standardize the point cloud size and augment the data.

For the custom dataset (e.g., RealSense), use the `dataloader_custom.py`

2. Training the Model

The training script is contained in `train.py` and performs the following steps:

- **Model Architecture:** The model used is `DGCNNClassifier`, defined in `model.py`. It implements the Dynamic Graph CNN architecture for point cloud classification.
- **Dataset Creation:** Processes training sequences specified in the script (e.g., MIT sequences like "mit_32_d507/d507_2", etc.) by calling `process_sequences` from the dataloader.
- **Data Augmentation:** Applies `random_augmentation` to increase data variance.
- **Cross-Validation:** Uses 5-fold cross-validation to split the training data into training and validation subsets.
- **Training Loop:** Trains the DGCNN model while logging training and validation loss/accuracy.
- **Best Model Selection:** The best model is chosen based on the highest validation accuracy. If more than one epoch achieves the same accuracy, the epoch with the lowest loss is selected.
- **Saving:** The best model state is saved in the `best_models/` directory, and the training curves are saved in the `figures/` directory.

To run training:

```
python src/pipelineA/train.py
```

3. Testing / Evaluation

Using MIT/Harvard Test Data

The `test.py` script evaluates the saved model on the Harvard test dataset. This script:

- Processes the test sequences.
- Loads the best model from `best_models/`.
- Computes evaluation metrics: loss, accuracy, confusion matrix, precision, recall, and F1-score.
- Saves visualizations such as the confusion matrix and test performance metrics in the `figures/` directory.

To run evaluation on the test dataset:

```
python src/pipelineA/test.py
```

Using a Custom (RealSense) Dataset

For custom datasets, the `test_custom.py` script is used. It:

- Loads your custom RealSense dataset using the corresponding dataloader.
- Runs predictions using the trained model.
- Overlays predicted labels on the raw test images and saves them (with labels on edges) for visual inspection.

To run custom dataset testing:

```
python src/pipelineA/test_custom.py
```

Model Architecture

The `DGCNNClassifier` uses:

- Dynamic Graph Construction based on k-nearest neighbors
- Edge Convolution Blocks to capture local geometric features
- Global and local feature aggregation
- MLP classifier head

Pipeline B – Depth Estimation

This pipeline focuses on estimating depth from RGB images, which can then be used for table detection in Pipeline A or segmentation in Pipeline C.

Important Note: For this pipeline to work, please ensure the relevant predicted `depth_pred` folders are present in each folder. They have been provided in this submission and can also be generated by running the script below(this does take around 10-15 mins tho).

Features

- **Depth Estimation**: Generates depth images from RGB using pretrained models

- **Evaluation**: Computes depth accuracy metrics against ground truth
- **Preprocessing**: Prepares estimated depth maps for model input

How to Run

Depth Estimation for MIT/Harvard Datasets

```
python src/pipelineB/depth.py --folders_type mit # choices = mit/harvard
```

This will:

1. Process RGB images from the specified dataset folders
2. Generate depth maps using ZoeDepth pretrained model
3. Save depth maps as .npy files and visualizations as .png files
4. Compute evaluation metrics if ground truth is available

Depth Estimation for Custom RealSense Data

```
python src/pipelineB/depth_custom.py --folders_type ucl
```

This will:

1. Process RGB images from your RealSense dataset
2. Generate depth maps and save them in the appropriate format
3. Evaluate predictions if ground truth depth is available

Testing with Estimated Depth Maps

```
python src/pipelineB/test.py        # For Harvard/MIT datasets
python src/pipelineB/test_custom.py # For custom RealSense dataset
```

Implementation Details

- The depth estimation uses the ZoeDepth model from the Transformers library
- Generated depth maps are saved in each dataset's folder under `depth_pred`
- Evaluation metrics include AbsRel, RMSE, MAE, and Log10 error measures
- Visualizations show ground truth vs. predicted depths where available

Pipeline C – Point Cloud Segmentation Using DGCNN

This pipeline performs pixel-level segmentation to identify which points in a point cloud belong to a table versus background.

Features

- Point-wise segmentation labels (table vs. background)
- Training on polygon-annotated data
- Class-balanced downsampling to handle imbalance

How to Run

Training the Segmentation Model

```
python src/pipelineC/train.py --batch_size 16 --epochs 50 --lr 0.001 --save_dir src/pipelineC/checkpoints
```

Key parameters:

- `--batch_size`: Samples per batch
- `--epochs`: Number of training epochs
- `--lr`: Learning rate
- `--table_weight`: Weight for table class in loss function (default: 2.0)

Evaluation on Harvard Datasets

```
python src/pipelineC/evaluate.py --model_path src/pipelineC/checkpoints/dgcnn_seg_best.pth --visualize
```

This evaluates the model across four Harvard sequences, generating metrics and visualizations. The predictions can be found in `results/visualizations` folder.

Inference on Custom Data

```
python src/pipelineC/inference.py --model_path src/pipelineC/checkpoints/dgcnn_seg_best.pth --data_dir data/RealSense/UCL_Data1
```

This will:

1. Process depth images from the specified directory
2. Generate segmentation predictions
3. Save visualizations and statistics in `results/custom_inference` folder

Model Architecture

The segmentation model uses a DGCNN architecture:

1. **Edge Convolution Blocks**: Learn local geometric features using k-nearest neighbors

2. **Feature Aggregation**: Combine features from different levels via skip connections
3. **Point-wise MLP**: Produce per-point segmentation labels (table vs. background)

Evaluation Metrics

- **IoU (Intersection over Union)**: Measures overlap between predicted and ground truth segments
- **Accuracy**: Overall point-wise classification accuracy
- **Precision/Recall/F1**: Evaluates detection quality, especially for table points