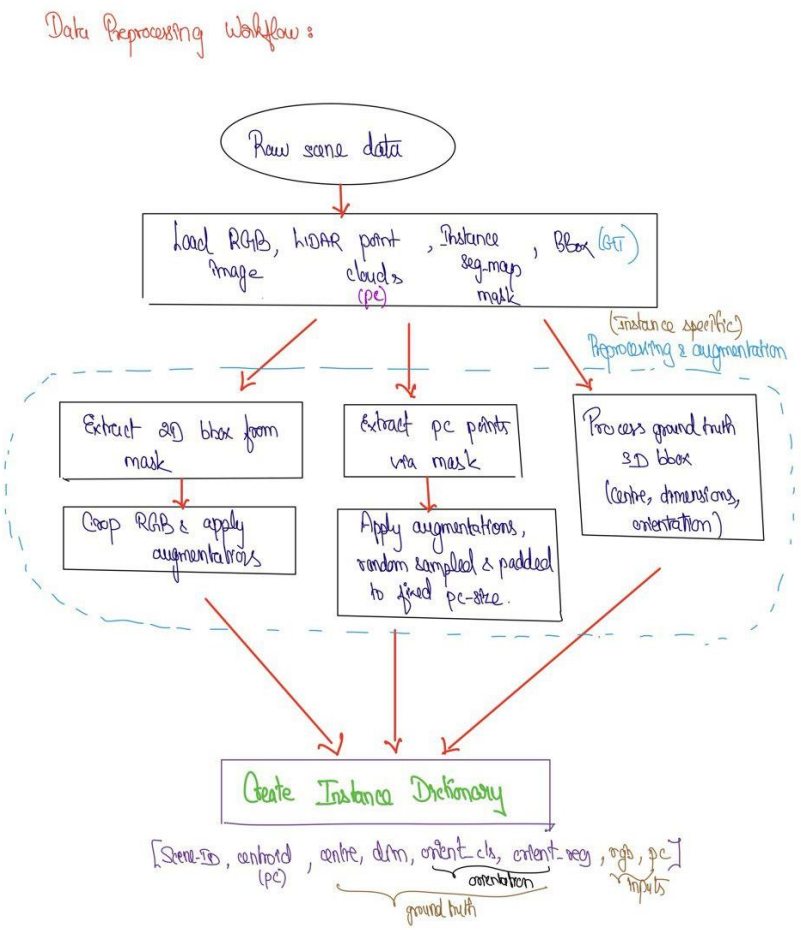
PROJECT: MULTI-MODAL BOUNDING BOX REGRESSOR

Below is a detailed, step-by-step analysis and documentation of the end-to-end deep learning pipeline for 3D bounding box prediction. In the sections below, I explain each important component, how the design decisions help solve the task, and how the various modules integrate into the overall system.

1.Data Augmentation

- RGB Transformations:
    - Applies resizing, normalization, and tensor conversion using Albumentations.
    - Adds random flips and color jittering during training.
- LiDAR Transformations:
    - Uses Kornia for 3D rotations and affine transforms.
    - Applied to training data only.

2. Dataset Loading and Preprocessing



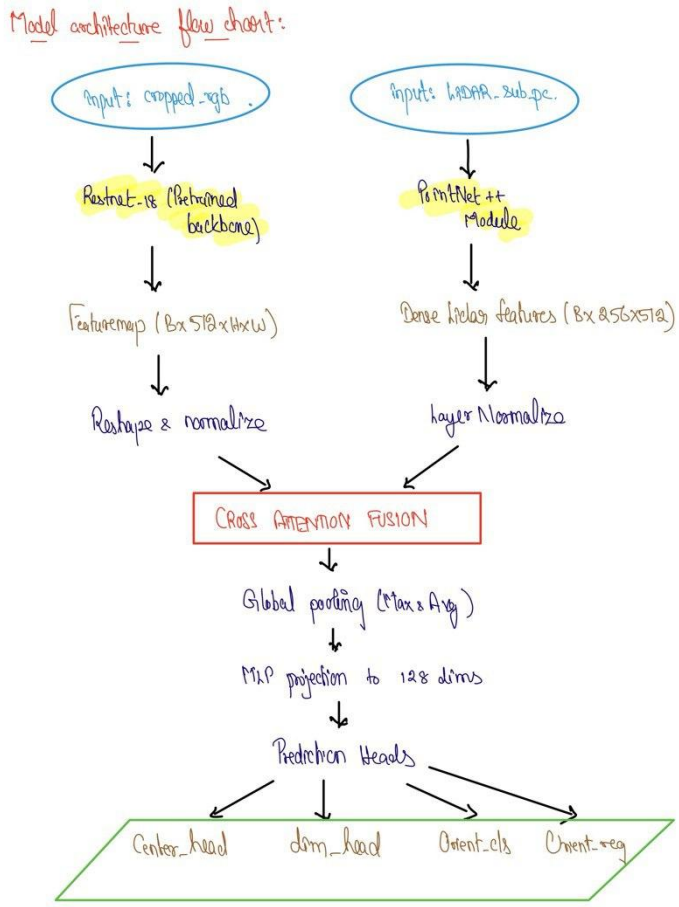
- Scene-level Loading:
    - Loads and validates scene data (RGB image, point cloud, segmentation mask, 3D bounding box).
- Per-instance processing strategy:
    - Mask and Bounding Box Extraction:
        - Extracts a 2D bounding box from the segmentation mask. This is used to crop the RGB image to the object region.
    - RGB Processing:
        - The cropped region is augmented using the previously defined RGB transforms.
    - LiDAR Processing:
        - The point corresponding to the mask are extracted.
        - A centroid is computed for the object, and the LiDAR points are either padded or randomly sampled to ensure a fixed size (512 points)
    - 3D GT Bounding Box Processing:
        - Computes the center, dimensions, and orientation (yaw) of the box.
        - The yaw angle is quantized into bins (multi-bin representation) and the residual is computed.

3. Data Loader and Batch Collation

- Custom Collate Function:
    - Because dataset returns a dictionary for each instance (including scene IDs, RGB crops, LiDAR tensors, centroids, and bbox data), a custom collate function is necessary to merge individual samples into a batch.
    - Ensures that the batched output contains separate keys for each predicted element (e.g., center, dims, orientation).
- Data Loader Setup:
    - Uses a train-test split (via scikit-learn's train_test_split) to divide the scenes into training, validation, and test sets.
    - Constructs PyTorch DataLoader objects for each split with proper batch size and shuffling.

4. Complete Model Architecture – Design choice

Model is built from several modular components that process each modality separately before fusing the information.



A. PointCloud Pathway – SimplePointNet++

- PointNet++ Inspired: Uses farthest point sampling (FPS) and radius-based neighborhood searches to progressively downsample and aggregate point features.
- Two stages are implemented using custom PointNetConv layers.
- Each stage first downsamples the point cloud (using FPS) and then builds local neighborhoods (using a radius query) before applying a convolution-like operation.
- Batch-wise input normalization
- Utilized methods and functions from lib: torch_geometric – PointNetConv, fps, radius

Why This Approach?

- **Sparse data handling:** PointNet++ is designed to work with unordered and sparse point cloud data. It uses farthest point sampling (FPS) and local neighborhood aggregation (via ball queries) to capture local geometric structures.
- **Feature Abstraction:** Design (downsampling from 1024 to 256 points with two FPS stages and local ball queries with increasing radii) ensures that local context is captured effectively while reducing computational load.

B. Image Pathway – ResNetBackbone

- ResNet-18 Backbone: Utilizes a pre-trained ResNet-18 for extracting high-level image features.
- The network is split into a stem (initial convolution, batch norm, ReLU, maxpool) and sequential residual layers.

- Early layers are frozen to retain general low-level features while fine-tuning later layers for the specific task.

Why This Approach?

- Transfer Learning: Using a light weight pre-trained network leverages existing learned representations, speeding up convergence.
- Acts as an auxiliary image modality branch to refine 3D Bbox predictions.

C. Fusion Mechanism – CrossAttentionFusion

- Cross-Attention: Uses a multi-head attention module where the LiDAR features act as the query and the image features act as key/value.
- Global features are extracted using both max and average pooling, followed by an MLP projection to merge the pooled features.

Why This Approach?

- Feature Alignment: By projecting image features into the same dimensional space as the LiDAR features and using cross-attention, the network can attend to the most relevant parts in each modality. This allows the fusion module to effectively integrate spatial cues from both the image and point cloud domains.
- Dynamic Weighting: The use of multihead attention helps the model to learn which regions of the image correspond to which parts of the point cloud, providing a dynamic way of fusing the two data types.
- Combining avg and max pooling to preserve more information, noting that fused feature values are generally sparse, leading to broader ranges in predicted dimensions.

D. Prediction Heads

- Center Prediction: A small MLP outputs the predicted center coordinates.
- Dimension Prediction: An MLP predicts the dimensions; a scale-factor parameter ensures the predictions start from a realistic scale.
- Orientation Prediction: Uses a multi-bin approach with separate branches for classification (to determine the bin) and regression (to predict the residual within that bin).

Why This Design?

- Multi-Modal Integration: The design explicitly separates processing of image and LiDAR data, then fuses them in a way that allows each modality to contribute complementary information.

5. Training Infrastructure

- Optimizer and Learning Rate Scheduler:
    - Uses AdamW with parameter-specific learning rates.
    - Dynamic Learning Rate Scheduler.
- Loss Computation:
    - Multi-Task Loss: Smooth L1 loss is used for regression tasks (center and dimensions), and a multi-bin approach is used for orientation (combining cross-entropy and smooth L1 loss). This is motivated by stability and handling angle periodicity.

Why multi-bin orientation loss?

- Handles periodicity: By discretizing the angle space, the network avoids discontinuities near the wrap–around.
- The combined classification–regression provided more stable gradients than a direct angular MSE. That's why switched from (sinθ,cosθ) continuous value representation after experimentation.

- Regularization Improvements:
  - Dropout has been added to force the network to learn more robust features that do not rely on specific activations.
  - Dynamic Learning Rate Scheduler: The scheduler monitors the validation loss and reduces the learning rate by a factor of 0.5 if the validation loss does not improve for defined number of consecutive epochs. This dynamic adjustment helps the model converge to a more general solution when training plateaus.
  - The training loop includes an early stopping mechanism. If the validation loss does not improve for a set number of epochs, training is stopped early to prevent overfitting.
- Training Loop:
  - Iterates over batches, moves data to GPU, and performs forward and backward passes.
  - Includes model validation during training and detailed evaluation.
  - Includes gradient clipping to avoid exploding gradients.
  - Logs various debug metrics and visualization (via TensorBoard)

6. Evaluation Metrics, Visualization and Logging

- Computes mean absolute error (MAE) and mean squared error (MSE) for both center and dimensions, while computes mean angle error for orientation.
- MAE and MSE provide insight into how well the model predict spatial parameters, while angular error and 3D IoU provide a holistic measure of box quality.
- Debugging visualizations for a single sample (augmented RGB image with its 2D bounding box and masked 3D point cloud) are logged only when Config.DEBUG_MODE is True.
- All global settings (e.g., data splits, image/LiDAR sizes, augmentation parameters) are contained in the Config class. Logging is configured to output messages to logging.txt.

7. Final Comments and Refinements

- Based on the log output, I observe that:
  - Training loss steadily decreases.
  - Validation loss decreases initially, but after a few epochs, the validation loss starts increasing even though the training loss keeps decreasing- Sign of overfitting
    Reason: May be due to limited data availability
  - High Mean Angle Error: still a major issue that needs addressing. Could be focussed on improving the yaw head architecture and loss formulation to improve the orientation predictions into a range that results in better 3D IoU. This may be due to high overlapping and dense objects.

- Summary: For each object instance I extract two corresponding inputs:
    - A cropped RGB patch from the full image (focused on a single object), and
    - A corresponding LiDAR sub-point cloud extracted from the scene's point cloud for that object.

  Each element in batch represents one object instance. After processing the two modalities with separate backbones (PointNet++ for the LiDAR and ResNet for the image), I fuse their features and use these fused features to predict a 3D bounding box. In post-processing you plan to group these object predictions into scenes using scene IDs.

  This approach falls into Instance-Based Refinement category, where we run an instance-level network to regress a refined 3D bounding box. It is like many two-stage object detection pipelines where (a) region proposals or object crops are generated and (b) a refinement network predicts the final box parameters. So basically, I worked on problem (b).
- Due to time constraints, I am not able to further refine the model and continue with Inference optimization.

Code Explanation

1. Global Configuration & Logging:
    - The Config class centralizes hyperparameters.
    - The logging is set to output important information to logging.txt.
2. Visualization Integration:
    - The DebugVisualizer class provides methods to log images and point clouds after data preprocessing.
    - In the dataset's _debug_visualization method, if Config.DEBUG_MODE is enabled, one sample is visualized via debugger.log_image.
3. Data Augmentation & Preprocessing:
    - The Augmentor and FrustumDataset classes load, verify, and preprocess the data while applying augmentations.
4. Model Architecture:
    - The network is composed of a LiDAR branch (simplified PointNet++), an image branch (ResNet-18), and a fusion module using cross-attention.
    - Prediction heads output center, dimensions (scaled), and multi-bin orientation predictions.
5. Training Infrastructure:
    - The BBoxTrainer class manages the training loop with optimizer, scheduler, and loss computation.
    - Essential loss and training information are logged.
6. Evaluation Metrics:

- The BBoxMetrics class offers a detailed evaluation—tracking errors for the center, dimensions, and orientation, and computing an overall 3D IoU (a reliable method from pytorch3D).
- Utility functions convert the model's output into 3D box coordinates.

References Used:

Pointnet++:

https://arxiv.org/pdf/1706.02413

Frustum PointNets for 3D Object Detection from RGB-D Data:

https://arxiv.org/pdf/1711.08488v2

PointFusion: Deep Sensor Fusion for 3D Bounding Box Estimation:

https://arxiv.org/pdf/1711.10871v2

PointNet:

https://arxiv.org/pdf/1612.00593

Multibinloss: 3D Bounding Box Estimation Using Deep Learning and Geometry

https://arxiv.org/pdf/1612.00496