Catharin Nivitha

Floorplan Segmentation and Bedroom Mapping

1. Project Overview:

This project focuses on building a model to segment and classify key elements in 3D bedroom floorplans using Mask R-CNN. The goal is to automate the extraction of the bedroom layout and segment important elements such as the bed, wardrobe, dresser, and loft. Additionally, the model classifies wardrobes into types: sliding parallel or walk-in.

2. Dataset Description:

The dataset comprises 100 annotated 3D bedroom floorplans collected from Pinterest in JPG format. Each image is manually annotated using Roboflow with bounding boxes and polygons to label furniture and room elements, including the bed, wardrobe, dresser, and loft. Wardrobes are further classified into three types: sliding, walk-in, and parallel.

Through augmentation, the dataset was expanded to 364 images, with transformations such as horizontal flips, 90° clockwise and counterclockwise rotations, cropping (0%–20% zoom), rotations (±15°), shearing (±10° horizontal and vertical), and adjustments to hue (±15°), saturation (±25%), brightness (±15%), and exposure (±10%).

Preprocessing steps included resizing the images, normalizing pixel values, and splitting the data into training and validation sets. The dataset is exported in COCO format, with each JPG image accompanied by its corresponding JSON annotation file.

3. Model Architecture:

I chose **Mask R-CNN** for this project because it is a state-of-the-art architecture designed for **instance segmentation**, which is ideal for tasks where both **object detection** and **pixel-level segmentation** are needed. In this project, the goal is to segment and classify various objects in floorplan images, such as **bedrooms, beds, dressers, wardrobes**, etc., which require precise pixel-wise segmentation along with object detection.

Key Components of Mask R-CNN:

1. **Backbone (ResNet50)**:
    - **ResNet50** is used as the backbone network for feature extraction. ResNet50 is a convolutional neural network (CNN) that leverages **residual connections** to allow deeper networks without suffering from the vanishing gradient problem.
    - The backbone processes the input image to extract feature maps, which are then used for detecting objects and generating segmentation masks.
    - In this architecture, **ResNet50** provides a balance between speed and accuracy, making it well-suited for floorplan images that contain intricate object layouts.
2. **Region Proposal Network (RPN)**:

- The **RPN** generates potential bounding box proposals for each object in the image. It scans the feature map produced by the backbone and outputs regions that are likely to contain objects.
- The RPN works by sliding a small network over the feature map and generating **anchors** at various positions and scales. The network then classifies whether each anchor is an object or background and refines the anchor's bounding box.
- This step helps narrow down the areas of the image that are worth further processing, focusing the model's attention on relevant portions.

3. **RoI Align**:
   - Once object proposals are generated by the RPN, **RoI Align** is used to extract fixed-size feature maps for each region of interest (RoI) without losing spatial resolution.
   - RoI Align avoids the quantization error introduced by RoI pooling, which makes it more precise when mapping object regions to their feature maps.
4. **Fully Connected Layers (FC) & Classification Head**:
   - After RoI Align, the model uses fully connected layers to classify the objects in each proposed region. This is where the model decides the class of each object (e.g., **bed**, **wardrobe**).
   - The classification head outputs the class probabilities for each region.
5. **Bounding Box Regression**:
   - The bounding box head refines the location of each detected object by predicting the offsets to the original RPN proposals.
   - This ensures that the object bounding boxes are more accurate and aligned with the object boundaries.
6. **Mask Branch**:
   - The **Mask branch** is unique to **Mask R-CNN** and generates a pixel-wise binary mask for each detected object.
   - This branch takes the feature maps from RoI Align and processes them through several convolutional layers to produce a mask for each object.
   - The mask predicts which pixels belong to the object and which do not, enabling pixel-level segmentation of the objects within the bounding boxes.

Modifications:

- **Custom Dataset**: I used a custom dataset of floorplans with objects such as **bedrooms**, **beds**, **dressers**, and various **wardrobes**. The dataset is segmented into different regions and labeled accordingly.
- **Loss Function Adjustment**: I adapted the loss function to handle the specific classes present in the floorplans, ensuring proper learning for each object class (e.g., **bedroom**, **dresser**).
- **Learning Rate and Augmentation**: I fine-tuned the learning rate and employed image augmentation techniques (like flipping, rotation, and scaling) to increase the model's robustness to different floorplan layouts and perspectives.

Data Flow:

- The input to the Mask R-CNN model is an image from the dataset (e.g., a floorplan image).
- The **backbone network** (ResNet50) extracts hierarchical features from the image.
- These features are passed to the **RPN**, which generates object proposals.
- The **RoI Align** layer then ensures each proposal is mapped accurately to a fixed-size feature map.
- The model performs **classification** to label each object and uses **bounding box regression** to refine the locations.
- Finally, the **mask branch** generates a pixel-wise mask for each object, providing detailed segmentation that outlines the boundaries of each detected object.

Why Mask R-CNN?

- **Instance Segmentation**: Mask R-CNN excels at **instance segmentation**, which is crucial for this project as we need to identify multiple instances of objects (e.g., multiple wardrobes) in a single floorplan image.
- **Accuracy and Precision**: The ability to generate pixel-wise masks ensures high precision in floorplan segmentation, which is important for applications like automated floorplan analysis or virtual interior design.
- **Flexibility**: Mask R-CNN's architecture allows for easy integration of additional branches, such as adding new object classes or adapting to other types of segmentation tasks.

By using Mask R-CNN, the model is able to accurately detect and segment objects within floorplan images, providing both object localization (via bounding boxes) and pixel-level segmentation (via masks), which is the primary requirement of this project.

4. Steps Followed:

Document each key step in your project, from dataset collection to model training and evaluation. Use clear, concise language and break each step into sub-sections if necessary.

Data Preprocessing:

- Describe how the data was loaded, cleaned, and augmented.
- Mention any preprocessing techniques used, such as resizing, normalization, or data augmentation.

Model Training:

- Explain how the model was set up for training, including parameters like learning rate, batch size, number of epochs, etc.
- If you used transfer learning, mention how the model was initialized with pre-trained weights and fine-tuned.

Evaluation:

- Describe how the model's performance was evaluated (e.g., metrics like accuracy, Intersection over Union (IoU), etc.).
- Mention any visualizations or evaluation scripts used to assess performance.

5. Challenges Faced:

1. Sparse or incomplete annotations in the dataset led to difficulties in achieving consistent model performance during early training.
2. Loss oscillations across epochs made it challenging to assess whether the model was converging effectively.
3. High classification loss (mrcnn_class_loss) persisted throughout the early phases of training, indicating possible issues with data representation or model optimization.
4. Hardware constraints limited the ability to use larger batch sizes, affecting gradient stability and training speed.

6. Solutions Implemented:

1. **Sparse or Incomplete Annotations**:
   To address the lack of sufficient annotations in the dataset, I applied various data augmentation techniques such as rotation, scaling, flipping, and cropping. This effectively increased data diversity and helped the model generalize better to unseen examples.
2. **Loss Oscillations**:
   Observing the oscillations in loss values during training, I experimented with adjusting the learning rate. A combination of fine-tuning the learning rate and employing a learning rate scheduler was used to ensure smoother convergence over epochs. This also involved monitoring the training curve to identify the optimal parameters.
3. **High Classification Loss**:
   Analyzing the training process revealed a class imbalance in the dataset. To mitigate this, I implemented a weighted loss function, assigning higher weights to underrepresented classes, thus ensuring balanced learning across all categories.
4. **Hardware Constraints**:
   Due to limited hardware resources, I optimized the model by reducing its size and adjusting the batch size to fit within the GPU memory. This also involved implementing mixed precision training to improve computational efficiency while maintaining model accuracy.

7. Results and Discussion:

The performance of the model was evaluated using several key metrics, including **Mean Average Precision (mAP)** at different IoU thresholds, **Intersection over Union (IoU)**, and **Loss values** (e.g., classification loss and mask loss). The results highlight the effectiveness of the model in segmenting the objects of interest.

1. Key Performance Metrics:

- **Mean Average Precision (mAP) at IoU 0.5:** 0.45
- **Intersection over Union (IoU) at IoU 0.5:** 0.53
- **Classification Loss:** 0.12
- **Mask Loss:** 0.30

These metrics provide an indication of the model's accuracy in detecting and segmenting the objects.

2. Comparison with Baseline:

A comparison with the baseline model shows that the model achieved significant improvements in both mAP and IoU:

- **Baseline Model:** IoU = 0.40, mAP = 0.37
- **Trained Model:** IoU = 0.53, mAP = 0.45

This shows an improvement in both the model's ability to localize and classify objects more accurately, especially with the use of data augmentation techniques and fine-tuning the learning rate during training.

3. Loss Progression Over Epochs:

The following table summarizes the loss values during training, showing how both the classification loss and mask loss decreased as the model learned from the data.

Epoch	Classification Loss	Mask Loss
1	1.24	0.68
2	1.12	0.60
3	0.95	0.55
...	...	...
30	0.12	0.30

This indicates that the model improved significantly over the 30 epochs of training.

4. Visual Comparison:

The following figures demonstrate the model's performance in detecting and segmenting the objects in comparison to the ground truth annotations. These visualizations show the model's segmentation masks overlaid on the input images.

- **Image 1: Ground Truth vs. Predicted Segmentation (Example 1)**

- **Image 2: Ground Truth vs. Predicted Segmentation (Example 2)**

These examples show the model's accuracy in object detection and segmentation. In the first example, the model successfully segmented the object, while in the second example, some inaccuracies were observed due to factors like occlusion or complex textures.

5. Conclusion:

The model demonstrated strong performance on the validation set, achieving a **Mean Average Precision (mAP)** of 0.45 and an **IoU** of 0.53. This represents a significant improvement over the baseline model. However, there are still areas for improvement, particularly in reducing the number of false positives and handling complex object shapes and occlusions.

8. Future Improvements:

- **Dataset Diversity and Size**:
  The current dataset has limitations in terms of size and diversity, which might restrict the model's ability to generalize. In the future, collecting a larger and more varied dataset with richer annotations would improve the model's robustness.
- **Architectural Enhancements**:
  Exploring deeper architectures such as ResNet101 or EfficientNet, which are better at feature extraction, could lead to improved model accuracy and performance. Additionally, experimenting with ensemble methods might further enhance results.
- **Handling Class Imbalance**:
  While weighted loss functions were applied, future improvements could include advanced techniques like oversampling, synthetic data generation for minority classes, or adopting algorithms specifically designed to handle imbalanced datasets.
- **Real-time Inference**:
  Optimizing the model for deployment in real-time applications could be explored, such as model pruning, quantization, and hardware acceleration, ensuring faster inference while maintaining accuracy.
- **Advanced Regularization Techniques**:
  Incorporating more sophisticated regularization methods, such as adversarial training or self-supervised pretraining, could help the model learn richer representations and reduce overfitting.
- **Hyperparameter Tuning**:
  While initial tuning yielded improvements, further exploration of hyperparameter

optimization techniques (e.g., Bayesian optimization or grid search) could uncover better settings for enhanced performance.

- **Integration of External Data Sources**:
  Adding complementary data sources, such as pre-trained embeddings or domain-specific datasets, could improve feature representation and model accuracy.

9. Code Walkthrough:

The project's code and resources are organized into the following components:

1. **Training Notebook (train.ipynb)**:
   - Handles the training pipeline, including:
     - Loading the dataset and annotations.
     - Applying data augmentation for better generalization.
     - Defining and setting up the Mask R-CNN architecture using TensorFlow 2.
     - Configuring hyperparameters like learning rate, batch size, and epochs.
     - Logging training progress and saving model checkpoints at regular intervals.
2. **Testing and Visualization Notebook (test.ipynb)**:
   - Focuses on evaluating and visualizing the model's performance:
     - Loading the trained model weights.
     - Running predictions on test images and visualizing results.
     - Overlaying predicted masks and bounding boxes on the original images.
     - Calculating performance metrics such as IoU, precision, and recall.
3. **Logs**:
   - Includes detailed training logs, such as loss values (RPN loss, classification loss, mask loss) across epochs.
   - Helps in monitoring model performance and debugging issues during training.
4. **Mask R-CNN TensorFlow 2 Model**:
   - The downloaded implementation of Mask R-CNN is used as the base for this project, adapted and fine-tuned to suit the dataset.
5. **Annotations**:
   - Contains the labeled dataset, including bounding boxes and segmentation masks.
   - Used during training to supervise the learning process and validate model predictions.

This structure ensures a streamlined workflow, with dedicated sections for training, evaluation, and logging, making it easy to navigate and build upon the project.

10. Conclusion:

This project successfully demonstrates the training and evaluation of a Mask R-CNN model for segmenting objects in the provided dataset. Key accomplishments include the effective handling of

complex annotations, achieving significant loss reduction during training, and generating accurate segmentation results on the test set. The structured workflow, incorporating data augmentation and proper model fine-tuning, was instrumental in improving the model's performance.

Challenges like fluctuating loss values and sparse annotations were addressed through methodical experimentation and careful parameter optimization, highlighting the importance of iterative problem-solving in machine learning projects.

Future steps include exploring additional strategies for improving model convergence, such as experimenting with different learning rates or optimizers, and expanding the dataset to include more diverse examples for better generalization. These enhancements will further increase the robustness and reliability of the model in real-world applications.

11. References:

- Aarohi Singla. "Mask R-CNN on Custom Dataset (2 Classes)." GitHub Repository. https://github.com/AarohiSingla/Mask-RCNN-on-Custom-Dataset-2classes-/
- Immersive Limit. "Using Mask R-CNN on Custom COCO-like Dataset." Tutorial. https://www.immersivelimit.com/tutorials/using-mask-r-cnn-on-custom-coco-like-dataset

These resources were instrumental in guiding the implementation of the Mask R-CNN model for custom dataset training and evaluation.