Visual Recognition Homework 2: Digit Recognition with Faster R-CNN

110550108

GitHub Repository: [Link]

1 Introduction

The goal of this homework is digit recognition using object detection techniques. The problem is divided into two tasks: (1) detecting the class and bounding box of each digit in an image, and (2) determining the number represented by the sequence of detected digits.

I approach this problem using Faster R-CNN, a two-stage object detection framework consisting of a backbone for feature extraction, a Region Proposal Network (RPN) for generating candidate object regions, and a head for classifying and refining bounding boxes.

To improve performance on this digit-specific task, I customized parts of the Faster R-CNN architecture, including modifying the anchor settings and post-processing steps. In addition, mixed-precision training and gradient accumulation were employed to speed up training and reduce memory usage.

2 Method

2.1 Data Preprocessing

The training dataset is in COCO format and includes bounding box annotations for digits. I implemented a custom `DigitDataset` class to load images and annotations. The test set lacks annotations and is loaded using a separate `TestDigitDataset` class. All images are transformed using standard ImageNet normalization and resized internally by the model.

2.2 Model Architecture

I use the `fasterrcnn_resnet50_fpn_v2` model from TorchVision as my base detector. This model includes:

- **Backbone**: A ResNet-50 architecture with Feature Pyramid Network (FPN), pre-trained on ImageNet.
- **Neck (Region Proposal Network)**:
  - Customized anchor generator with five scales: (16, 32, 64, 128, 256)
  - Three aspect ratios per scale: (0.25, 0.5, 1.0)
  - NMS threshold: 0.6
  - Proposal limits: `pre_nms_top_n_train = 1000`, `post_nms_top_n_train = 500`, `pre_nms_top_n_test = 500`, `post_nms_top_n_test = 250`
- **Head (RoI Head)**:
  - A customized box predictor with output for 11 classes (10 digits + background)
  - NMS threshold: 0.4
  - Maximum number of detections per image: 10

2.3 Training Settings

- Epochs: 5
- Batch Size: 32
- Gradient Accumulation: 4 steps
- Learning Rate: 1e-4
- Optimizer: AdamW with OneCycle learning rate scheduler
- Mixed Precision Training: Enabled via PyTorch AMP and GradScaler
- Device: CUDA (GPU)

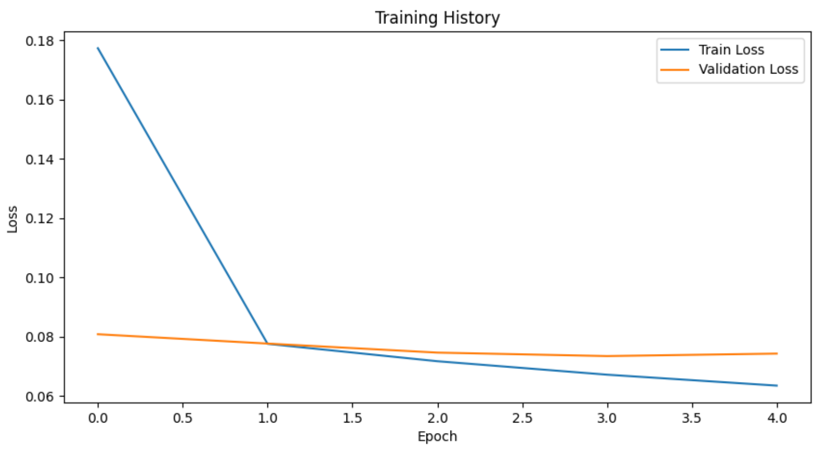
Post-processing for Task 2 involves:

- Filtering predictions with confidence > 0.7
- Sorting detections by x-axis
- Concatenating digits to form the final number

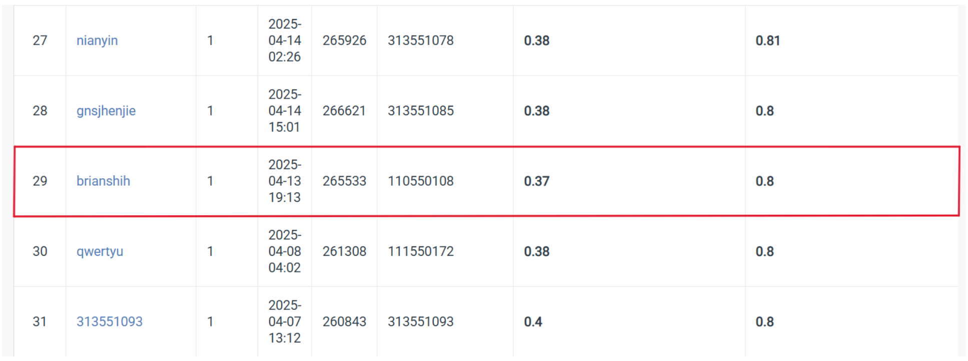
3 Results

3.1 Model Performance

The best validation loss achieved was **0.0771**. The following is the training and validation loss curves:



Public Leaderboard:

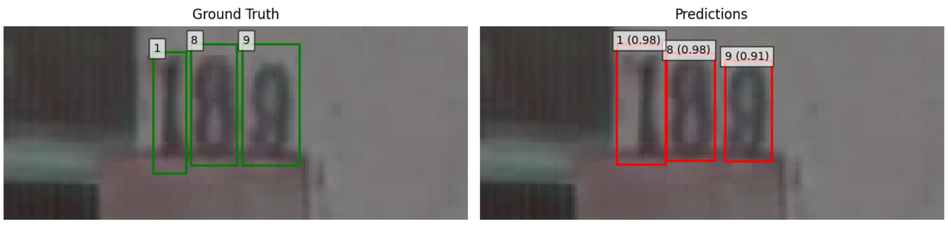


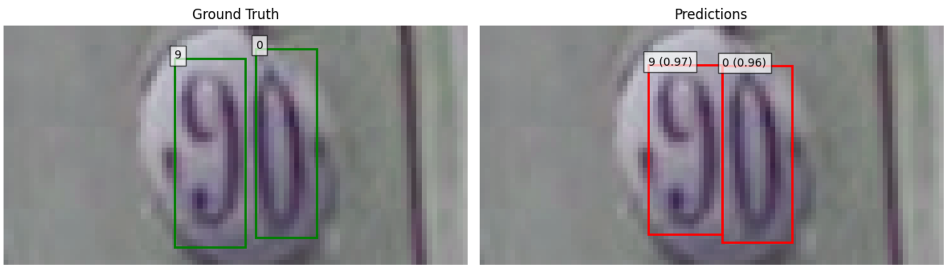
27	nianyin	1	2025-04-14 02:26	265926	313551078	**0.38**	**0.81**
28	gnsjhenjie	1	2025-04-14 15:01	266621	313551085	**0.38**	**0.8**
29	brianshih	1	2025-04-13 19:13	265533	110550108	**0.37**	**0.8**
30	qwertyu	1	2025-04-08 04:02	261308	111550172	**0.38**	**0.8**
31	313551093	1	2025-04-07 13:12	260843	313551093	**0.4**	**0.8**

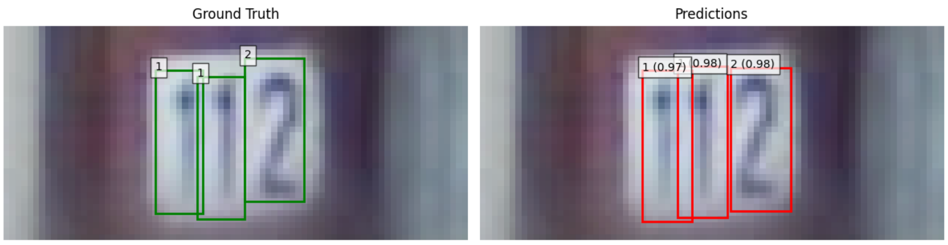
3.2 Key Findings

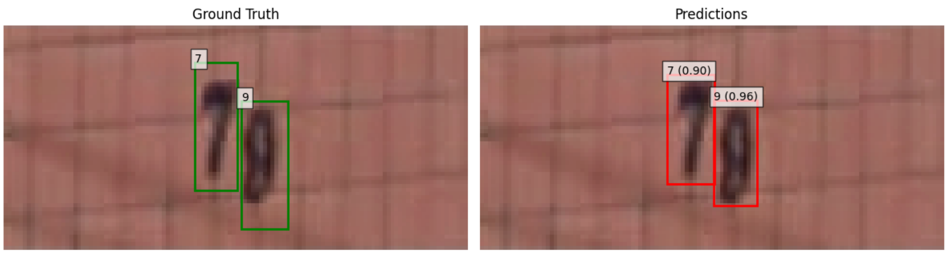
- The model quickly learned to detect digit locations and classifications. Mixed-precision training significantly reduced memory usage and increased training speed.

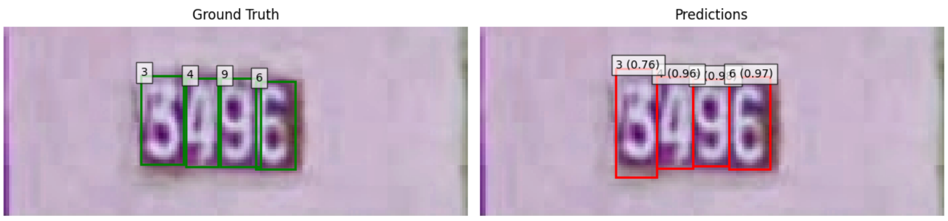
3.3 Predictions Visualization











The post-processing effectively solved Task 2 with high accuracy.

4 Additional Experiments: Comparisons of loss functions

4.1 Hypothesis

I hypothesize that using **GIoU loss** or **Smooth L1 loss with different beta values** could help improve the regression performance of the bounding box. The default Smooth L1 loss is effective but may not be optimal for digits, which typically occupy small and tightly packed regions.

4.2 Expected Outcomes

I expect that:

- GIoU loss will produce tighter and more accurate bounding boxes, particularly for overlapping digits.
- A smaller β in Smooth L1 loss will make the loss function behave closer to L1 loss, which may provide better gradient signals for small box offsets, improving localization precision.

4.3 Method

I compared the following settings:

1. **Baseline**: Default Faster R-CNN using Smooth L1 loss ($\beta = 1.0$)
2. **Experiment A**: Use GIoU loss for bounding box regression
3. **Experiment B**: Use Smooth L1 loss with smaller $\beta = 0.5$

4.4 Results & Analysis

Loss Function	**Validation loss**	**Qualitative Observations**
Smooth L1 ($\beta = 1.0$)	0.0795	Stable training, accurate boxes, some overlapping digits
GIoU Loss	0.0823	Slight improvement in box tightness, but less stable
Smooth L1 ($\beta = 0.5$)	0.0771	More precise boxes, slightly better digit ordering accuracy

Using a smaller β in Smooth L1 helped the model better localize small bounding boxes, improving digit order prediction. GIoU gave tighter boxes, but was harder to train and slightly less stable.

5 References

- Ren, Shaoqing, et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." *NeurIPS*, 2015. `https://arxiv.org/abs/1506.01497`
- TorchVision Faster R-CNN implementation: `https://github.com/pytorch/vision/blob/main/torchvision/models/detection/faster_rcnn.py`
- PyTorch Mixed Precision Training Guide: `https://pytorch.org/docs/stable/amp.html`
- Rezatofighi, H., et al. "Generalized Intersection over Union: A Metric and a Loss for Bounding Box Regression." *CVPR*, 2019. `https://arxiv.org/abs/1902.09630`