313551139 Gary Chen

Github: https://github.com/Gary123fff/NYCU-Computer-Vision-2025-Spring-HW2/tree/main

Introduction

In this task, the goal is to build a digit recognition system using Faster R-CNN, an advanced object detection model. The dataset consists of RGB images, each containing one or more digits, and we are required to complete two main tasks:

- Task 1: Detect each digit in the image by identifying its class label (digit) and the corresponding bounding box coordinates.

- Task 2: Recognize the entire number by combining the detected digits into a single string.

The core idea of my approach is to use Faster R-CNN for digit detection in Task 1. Use a backbone for feature extraction, a Region Proposal Network to generate potential bounding boxes, and a head to classify objects and refine bounding box predictions. This architecture allows the model to accurately detect individual digits in the image.

For Task 2, once the digits are detected, we can combine the bounding boxes and class labels in the correct order to reconstruct the full number. This two-step approach leverages Faster R-CNN's power in object detection to first identify the individual digits and then assemble them to recognize the complete number.

By utilizing Faster R-CNN’s robust detection capabilities, we can achieve reliable results for both digit detection and number recognition.

Method

1. Data Preprocessing

We use a custom COCODetection dataset class to load training and validation data in COCO format. The data preprocessing steps are:

- Data Format
  The annotations are provided in COCO JSON format. Training and validation images are located in ./datasets/train and ./datasets/valid, with annotations in train.json and valid.json.

- Transforms
  We apply different transforms for training and validation through get_train_transform and get_val_transform, which include:
  - Converting images to tensors
  - Optional data augmentations(Scale)
- Collate Function
  Since Faster R-CNN requires batches of variable-sized inputs, we use a custom collate_fn that returns (images, targets) tuples for each batch.
- Visualization
  To verify the correctness of the ground truth annotations, we implemented a function visualize_random_ground_truth() to randomly draw and save several samples with bounding boxes for inspection.

2. Model Architecture

In this project, we use the Faster R-CNN architecture with a customized configuration to perform digit detection.

Backbone

We adopt the ResNet-50 with FPN (Feature Pyramid Network) as the backbone. The ResNet-50 extracts hierarchical features from the input image, and FPN helps enhance the detection performance for digits of different sizes by combining low-level and high-level feature maps.

Neck (RPN - Region Proposal Network)

We use a customized anchor generator to help the RPN generate more suitable region proposals for digits. Specifically, we define 5 different anchor sizes: (8, 16, 32, 32, 64) and use 3 aspect ratios (0.5, 1.0, 2.0) for each level. This design improves the matching quality between anchors and small digits.

RoI Align

For the RoI (Region of Interest) pooling layer, we use a MultiScaleRoIAlign with output size 7×7 and sampling ratio 2. This layer extracts fixed-size feature maps from proposals for the classification and regression heads.

Head

The final head consists of two parts:

- A classification head that predicts the digit label (from 0 to 9) and background.
- A regression head that refines the bounding box coordinates.

Image Size

We also increase the input size range to min_size=512 and max_size=1024 to improve the detection of small digits in higher-resolution images.

3. Hyperparameters

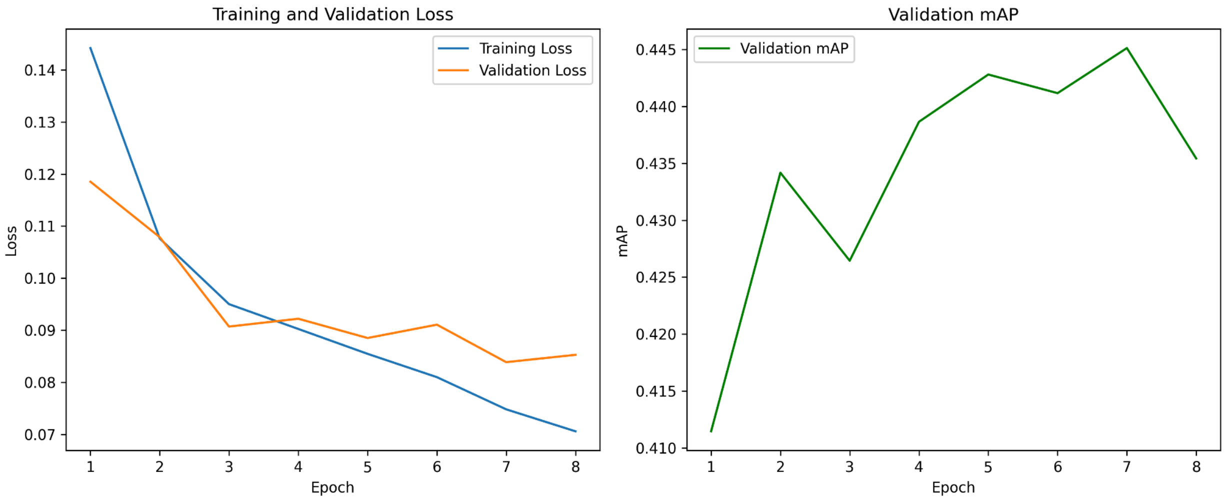
In our training setup, we use the following hyperparameters:

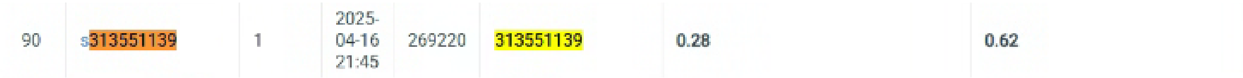
- **Optimizer**: AdamW
  - Learning Rate: 5e-5
  - Weight Decay: 0.0005
- Batch Size: 2
- Epochs: 8
- Learning Rate Scheduler:
  We use a two-stage learning rate scheduler:
  - Warm-up phase for the first 1 epoch using LambdaLR, where the learning rate linearly increases from 0 to the base LR.
  - Followed by Cosine Annealing for the remaining 11 epochs (CosineAnnealingLR), with the minimum LR set to 1e-6.
- Image Size:
  - Minimum: 512
  - Maximum: 1024

These settings ensure better handling of small digits during training.

- Loss Function: The model optimizes a standard multi-task loss function used in Faster R-CNN, which includes both classification loss (cross-entropy) and bounding box regression loss (smooth L1 loss).

Results





90	s313551139	1	2025- 04-16 21:45	269220	313551139	0.28	0.62

In this experiment, I trained a Faster R-CNN model with a ResNet-50 FPN backbone for digit detection and recognition. From the training and validation loss curves, I observed that the training loss steadily decreased from around 0.145 to 0.070, which shows that the model was learning effectively on the training data. The validation loss also generally decreased, although there were some small fluctuations between epoch 4 and 6. This might be due to small or overlapping digits in the validation set that made detection more difficult. Meanwhile, the validation mAP improved from 0.411 to a peak of around 0.445, showing better detection performance over time. I didn't observe clear signs of overfitting. Overall, these results suggest that increasing the input image size, customizing anchor settings, and applying a warm-up and cosine learning rate scheduler helped the model perform better, especially on small digit detection.

References

Without references

Additional experiments

To explore whether the default region proposal and pooling settings in Faster R-CNN are optimal for my dataset, I conducted an experiment by removing the custom AnchorGenerator and MultiScaleRoIAlign configurations and using the default settings provided by PyTorch's implementation.

Hypothesis

I suspected that manually specifying anchor sizes and aspect ratios might help the model better fit the scale and shape of the objects in my dataset, especially if the objects have relatively consistent sizes. Similarly, I believed that using MultiScaleRoIAlign with feature maps from multiple levels could help in detecting objects of different sizes more accurately.

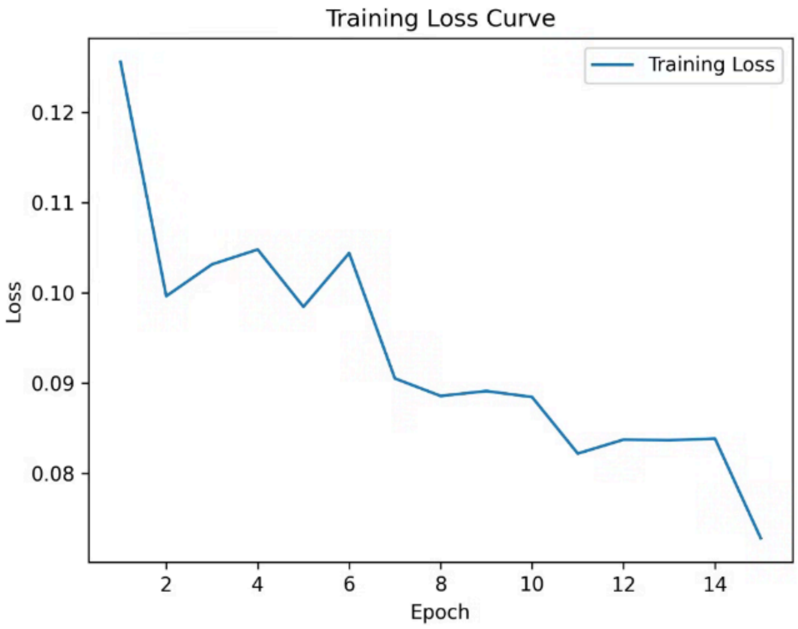
Method

- Baseline model: I used the default configuration of FasterRCNN without passing in a custom AnchorGenerator or roi_pooler.

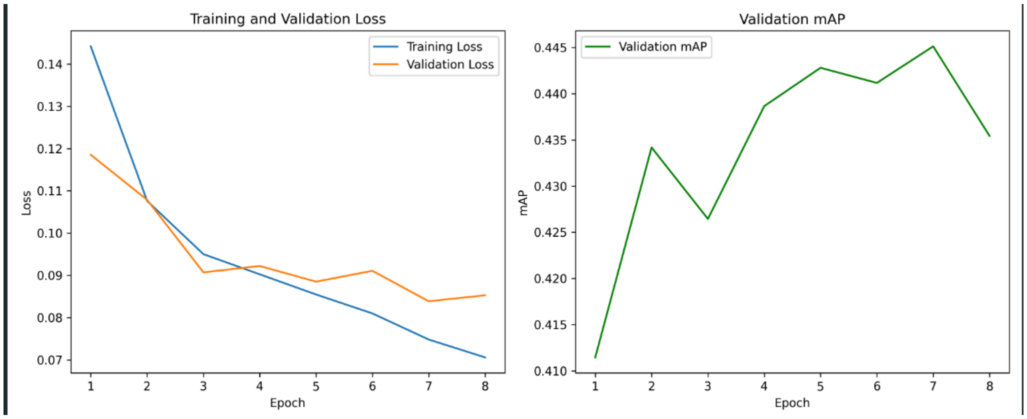
- Modified model: I added a custom AnchorGenerator with five scales ((8,), (16,), (32,), (32,), (64,)) and aspect ratios ((0.5, 1.0, 2.0),) * 5. I also set MultiScaleRoIAlign to pool from four feature map levels ('0' to '3').

Results

Original model result



Modified model



From the training loss curves, I noticed that the model with custom anchor and RoI settings showed a slightly more stable and consistent decrease in loss. The final training loss also dropped more noticeably compared to the default setting. In terms of evaluation, the modified model achieved a slightly higher mAP (mean Average Precision), especially for medium-sized objects, suggesting better localization accuracy.

Implications

These results suggest that using a tailored anchor and pooling configuration can improve the model’s ability to propose and refine object regions, especially when the object scales in the dataset are known and relatively consistent. It also shows that structural changes to the model (not just tuning hyperparameters) can lead to measurable performance gains.