Visual Recognition using Deep Learning HW2 – Report

313561002 梁巍濤

Table of Content

Introduction

The second homework is a Digit Recognition Problem, which is a classic machine learning task. For this implementation, we are asked to do two tasks: a class and bounding box for each digit that we detect inside the image, and a prediction of the digits itself inside of the given image. For this homework we are asked to only use Faster R-CNN as the model, with modifications allowed to each part. There is a total of 33,402 RGB images and JSON files in COCO format for training and validation, and a total of 13,068 images to be tested for prediction.

Since this homework directly asks us to use Faster R-CNN to tackle the problem, I will be implementing my idea by applying ResNet101-FPN as the backbone for this digit detection problem. My implementation will also try and improve upon the standard Faster R-CNN model through customized anchor generation and post-processing techniques for digit recognition.

Method

1. **Environment Setup**

   In this homework I used the following environment to try and implement what is asked:
   Hardware: NVIDIA Geforce RTX 4090
   Software:

   - OS: Ubuntu 20.04.6 LTS
   - Python: 3.10.15
   - Pytorch: 2.5.1+cu124
   - Conda (anaconda3)

2. **Data Pre-processing**

   Homework 2's dataset is split into images and JSON files for training, validation and testing. The images are applied pre-processing in the form of conversion to RGB format to handle grayscale or other color spaces, then normalized by using ImageNet mean (0.485, 0.456, 0.406) and standard deviation (0.229, 0.224, 0.225), and finally it gets randomly flipped, and gets contrast and brightness adjustment for more variety in the training/validation dataset.

3. **Model Architecture**

   The main backbone for the model I'm currently using is the ResNet101-FPN (feature pyramid network). Since pretrained weights are allowed I also use it from ImageNet for better feature extraction. FPN also provides multi-scale feature maps to detect objects of different sizes. The Neck (RPN) is described as follows for my implementation: Custom anchor generation with 5 different sizes (16, 32, 64, 128, 256), Three aspect ratios (0.5, 1.0, 2.0) for each anchor size to handle digit variations, Sliding window approach over feature maps to generate proposals, Classification branch to identify object/non-object regions, Regression branch to refine anchor boxes, and NMS (Non-Maximum Suppression) with threshold 0.5 to reduce overlapping proposals.

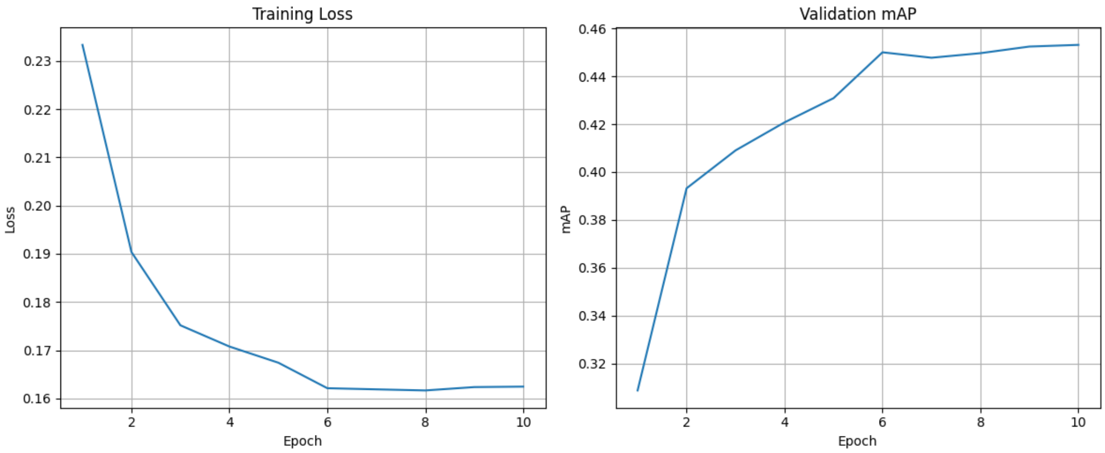
   The head of the Faster R-CNN uses Region of Interest pooling to extract fixed-size features from proposals and has two parallel branches: one is a classification branch for the digit classes and background, and another for the bounding box regression for refining the coordinates. Box scores are held to a threshold of 0.05 in order to filter the low-confidence detections, and there is a minimum size of 600 and maximum size of 1000 for the input images.

4. **Training Details**

   Using the method I had from my first homework, I also use a checkpoint system in this task's implementation, this is because of how extensive the time is needed to train the model for each epoch. The average training time I had for each epoch is between 10~15 minutes and I usually train it for 20 epochs, so it can take up to 4~5 hours just to find out whether my model was okay or not. I also use an optimizer to keep track of the training hyperparameters and use an NMS threshold and Confidence threshold in attempt to keep the model so that it would not stoop lower. Post processing is applied in my implementation to convert the detected digits into the correct numbers within the correct format.

Results

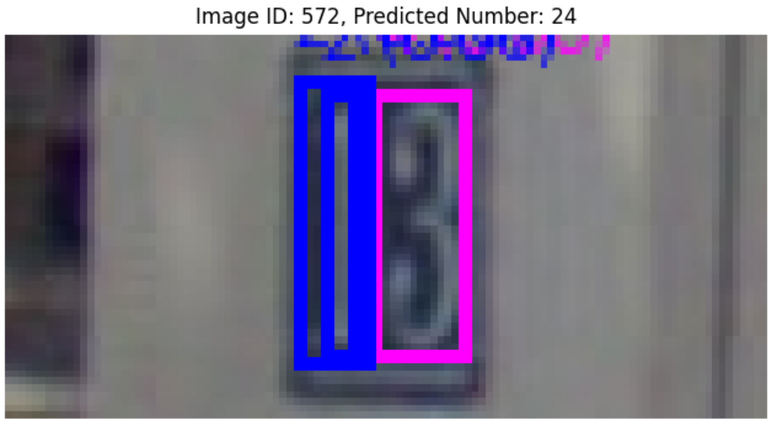
1. **Training loss, Validation mAP**



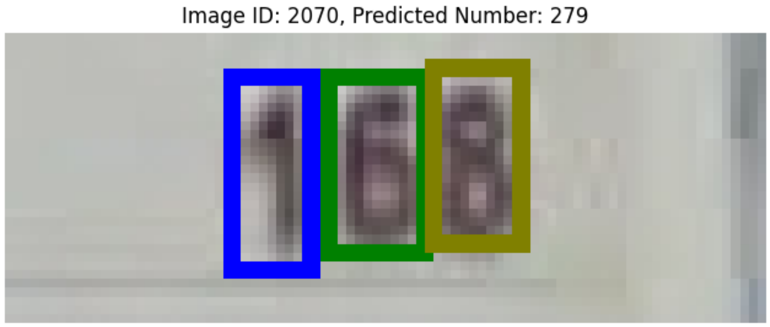
This is a shortened version of the training loss and validation mAP score I got from the first 10 epochs running the model. Even from this graph we can see that the training loss eventually plateau's as early as around Epoch 6, while validation mAP can achieve higher scores here than when I submit the implementation itself to codabench, which might be because of the difference of my evaluation function for both training accuracy and validation mAP score.

2. **Visualizations**

Example Visualization at Epoch 5:



Example Visualization at Epoch 20:



Should note that the example predicted numbers are referring to the category_id's in my code, so that's why the predicted numbers are 1 incremental above each digit detected (should be 1-6-8 but it prints out 2-7-9, small mistake that's fixed in the prediction output file later in the code).

3. Future Recommendations

I experienced a lot of scores that got plateau'd in the very early epochs, but I couldn't really do much with my implementation other than wait for the training to finish and hope that it could bring a better result. Seeing the other classmates achieve a Task 2 score of above 0.8 made me feel like my model could find a lot of room for improvement, although I am quite happy with my Task 1 score.

Additional experiments

1. **Anchor Configuration optimization**

   Why: Standard anchor sizes may not be optimal for digit detection; a custom configuration with smaller anchors (16-256) and specific aspect ratios (0.5, 1.0, 2.0) would better match the characteristics of digit objects.

   How: Replace anchor sizes to 16, 32, 64, 128, and 256 and change aspect ratios to ((0.5, 1.0, 2.0),) * len(anchor_sizes)

   Results: Changing anchor sizes to better suit the task needed made the model collect better results and improve the digit detection performance, especially for varying digit sizes.

2. **Custom Threshold Optimization**

   Why: Using different thresholds for different tasks should be implemented to optimize the result of different tasks. In this experiment I applied a threshold to detect the bounding box scores and for the relevant detections later for the whole number recognition.

   How: Added a 0.05 threshold for initial detection for the box_score_thresh which is a lower threshold but then added a higher threshold later with a value of 0.5 so that the digit detection could be more confident and have better results overall.

   Results: After trying several different values for the two threshold scores, I find that giving a lower initial threshold at 0.05 and a higher value at 0.5 later gave me the best result I could get. It got a few additions to the score of my submission even though not much.

References

- Ren, S., et al. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks
- Lin, T., et al. (2017). Feature Pyramid Networks for Object Detection

- He, K., et al. (2016). "Deep Residual Learning for Image Recognition"
- ResNet architecture by Torchvision