HomeWork 2

Faster R-CNN Model

Digit Recognition

國立陽明交通大學

NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Prepared by

110550124 林家甫

1 INTRODUCTION

The core objective of this homework is to use the Faster R-CNN machine learning model to detect digits (0–9) in the given images. Based on the coordinates of the detected digits, we then recognize the full number present in each image. The dataset consists of 30,062 training images, 3,340 validation images, and 13,608 test images.The data is provided in the COCO format, which is a standard format commonly used for object detection tasks.

This homework consists of two main tasks: (1) detecting all the digits in an image, and (2) recognizing the full number by analyzing the spatial arrangement of the detected digits. To tackle these tasks, I used PyTorch's built-in model fasterrcnn_resnet50_fpn_v2. In fact, I experimented with both fasterrcnn_resnet50_fpn_v2 and fasterrcnn_resnet50_fpn, and will present the final results and comparisons in the Additional Experiments section.

Code Reliability: pep8
GitHub Link: https://github.com/chiafu2018/Digit-Recognition

2 METHOD

I used the **fasterrcnn_resnet50_fpn_v2** model to implement this task. Compared to object classification, object detection is more robust and computationally intensive. With the same GPU resources used in HW1, each epoch in this task takes approximately two to three times longer to complete.

- **Data Preprocessing**

For this digit recognition task, I did not apply any image transformations such as rotation or resizing. This is because such transformations would also require corresponding adjustments to the bounding box annotations, which can be cumbersome and error-prone to implement correctly.

Then, I used PyTorch's built-in CocoDetection class to load the training and validation datasets in COCO format. To be noticed, the batch size is kept small (4 for training/validation and 2 for testing) due to the computational complexity of object detection. Unlike simple classification tasks (HW1) where each input is resized into a fixed-size image with a single label, object detection involves multiple objects per image as well as dynamic-sized annotations (bounding boxes and labels).

```
train_transform = T.Compose([T.ToTensor()])
```

```
    test_transform = T.Compose([T.ToTensor()])

    train_dataset = CocoDetection(root="./nycu-hw2-data/train/",
annFile="./nycu-hw2-data/train.json", transform=train_transform)
    val_dataset = CocoDetection(root="./nycu-hw2-data/valid/",
annFile="./nycu-hw2-data/valid.json", transform=train_transform)
    test_dataset = CustomTestDataset(root="./nycu-hw2-data/test",
transform=test_transform)
```

- **Model Architecture**

The Faster R-CNN architecture is composed of three main components: the backbone, the neck (Region Proposal Network or RPN), and the head. The **backbone** is typically a convolutional neural network such as ResNet, which extracts feature maps from the input image. These feature maps are then passed to the **RPN**, which acts as the **neck** of the architecture. The RPN slides over the feature map and generates a set of region proposals by predicting objectness scores and bounding box coordinates for multiple anchors at each spatial location. These proposals are then refined and filtered based on their confidence scores. Finally, the **head** component takes these proposed regions and applies ROI Pooling (or ROI Align) to extract fixed-size feature representations, which are passed through fully connected layers to perform object classification and further bounding box regression. Together, these components enable Faster R-CNN to efficiently and accurately detect objects in an image.

Like I mentioned before, I use **fasterrcnn_resnet50_fpn_v2** with pretrained weights to be the training model. The structure is provided by pytorch library, and it contains approximately 43M parameters.

```
model = fasterrcnn_resnet50_fpn_v2(weights="COCO_V1")
```

- **Hyperparameters**

Using a smaller learning rate not only leads to better final performance but also accelerates convergence. I believe this is due to the small batch size of the dataset—since parameters update every 2 epochs, a smaller learning rate helps prevent overshooting optimal values. I use the **ReduceLROnPlateau** scheduler, which decreases the learning rate when the loss plateaus, allowing for more controlled fine-tuning.

Additionally, since I'm working with pretrained weights, I intentionally avoid setting a large learning rate to prevent disrupting the useful features already learned by the model. In

the early stages of training—specifically, before epoch 5—I apply the technique of freezing the backbone parameters. This helps retain the pretrained features during the initial training phase, allowing the newly added layers to adapt to the specific task first without affecting the pretrained backbone.

```
# Freeze backbone for fine tuning
if frozen and epoch >= FREEZE_BACK_BONE_EPOCHS:
      for param in model.backbone.parameters():
          param.requires_grad = True
      frozen = False
```

- **Validation Metrics:**

To evaluate the model, I calculated the mAP using IoU thresholds ranging from 0.5 to 0.95 with a step size of 0.05, following the standard COCO evaluation protocol. Over 15 training epochs, I monitored the validation mAP and saved the model that achieved the highest mAP.

- **Task1: Digit Detection**

The output bounding boxes are in the format [x, y, x + w, y + h], whereas the COCO evaluation expects the format [x, y, w, h]. Therefore, I iterated through all predicted bounding boxes and converted them to the required format before evaluation.

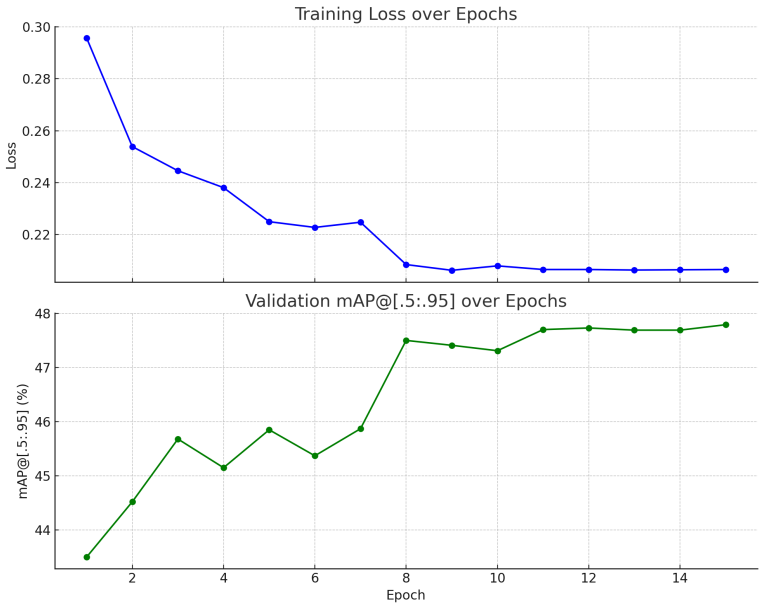
- **Task 2: Number Recognition**

To "recognize" the number in each image based on the digit detections from Task 1, I sorted the detected digits by their x-coordinate in ascending order, assuming that digits are always arranged left to right. If no bounding boxes are detected in an image, I output -1 to indicate the absence of numbers.

3 RESULTS

- **Task1: Digit Detection**

The highest validation mAP from 0.5 to 0.95 is around 0.47 , and the test mAP achieves 0.39 on Codabench. The following picture shows the average training loss (training loss per image) for 15 epochs.



- **Task 2: Number Recognition**

  The test score achieves 0.79 on Codabench.

4 REFERENCES

[1] Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks” [Online] Available: https://arxiv.org/pdf/1506.01497

5 ADDITIONAL EXPERIMENTS

- Different Faster R-CNN version Models - fasterrcnn_resnet50_fpn

  In the table below, I compare the performance of two built-in Faster R-CNN models. As shown, Faster R-CNN version 2 outperforms version 1 in terms of accuracy. However, a notable drawback of version 2 is that it has 2 million more parameters than version 1, which I believe slightly increases the computational requirements.

Model	mAP Score	Prediction Score
fasterrcnn_resnet50_fpn_v2	0.3826	0.7868
fasterrcnn_resnet50_fpn	0.3703	0.7104

- Sorting different x coordinates to recognize the number

In the table below, the digits are sorted based on their x-coordinates — specifically, the left edge, center point, and right edge — to achieve the final arrangement. As we can see, x coordinate at the left edge reached the highest prediction score.

X coordinate	Public	Private
Left	0.7868	0.7841
Middle	0.7856	0.7819
Right	0.7853	0.7824

Prediction Score