Selected Topics in Visual Recognition using Deep Learning, Homework 2

111550061, 邱冠崴

Part. 1, Introduction

In this assignment, our task is to use Faster R-CNN for digit recognition. We are required not only to detect the bounding boxes of digits (Task 1), but also to correctly recognize the full digit sequence within each image (Task 2). Initially, I selected Faster R-CNN with a MobileNet backbone from the torchvision library. However, I found that this model was too lightweight and tended to overfit the training data. As a result, I switched the backbone to ResNet-50, which led to improved performance — achieving a mean Average Precision (mAP) of 0.38 and an accuracy of 0.82 on the testing dataset.

Part. 2, Method

(1) Data Pre-processed:

In this assignment, we cannot apply data augmentation techniques as easily as we do in image classification tasks. For example, applying random horizontal flips or rotations requires modifying the corresponding ground-truth coordinates; otherwise, the predictions may become inaccurate. Additionally, the resolution of the images in this dataset is relatively low, so applying Gaussian blur could make the images difficult to interpret and potentially degrade model performance. Therefore, the only augmentation I applied was ColorJitter, to introduce some variability in color and brightness while preserving spatial information.

(2) Model architecture and hyperparameters:

Model	fasterrcnn_resnet50_fpn_v2 (in torchvision)
Optimizer	Adamw
Learning rate	$1*10^{-4}$
Weight decay	$1*10^{-3}$
Scheduler	StepLR (LR *0.5 every 2 epochs)

Model architecture (fasterrcnn_resnet50_fpn_v2):

The reason I chose this architecture is because it provides full pretrained weights specifically for object detection within the torchvision library. While deeper and more complex backbones such as ResNet-101 or ResNeXt-50 are available, their pretrained weights are typically for image classification tasks rather than object detection. In my opinion, using weights pretrained directly on object detection tasks leads to better performance than initializing with classification weights in backbone. Therefore, I selected this model as it is the most powerful and complex detection model in torchvision with fully pretrained object detection weights.

The main advantage of this model is that it is deep enough to capture complex and high-level features, which is crucial for accurate object detection. Its architecture allows it to learn rich representations that can generalize well across a variety of scenarios. However, a significant drawback is its high demand for computational resources—it requires a substantial amount of GPU memory and can be time-consuming to train and perform inference.

(a) backbone:

The backbone of this model is ResNet-50, which is a 50-layer deep convolutional neural network designed with residual connections. It is responsible for extracting high-level semantic features from input images. Additionally, the ResNet-50 is slightly modified to include FrozenBatchNorm2d instead of the standard BatchNorm2d. This means that the batch normalization layers do not update their statistics during training, which improves stability and performance, especially when training on smaller datasets or with smaller batch sizes.

(b) neck (RPN):

The neck component is the Feature Pyramid Network (FPN) v2, which enhances the backbone by generating a pyramid of multi-scale feature maps. FPN v2 combines high-resolution, low-level features with

low-resolution, high-level features using top-down and lateral connections. This allows the detector to be more effective at detecting objects of varying sizes. The v2 version in particular benefits from better normalization, improved efficiency, and tighter integration with the detection pipeline.

(c) head of Faster R-CNN:

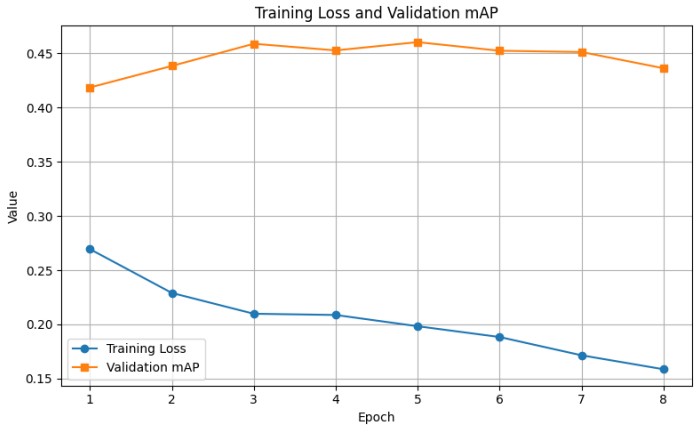
The head in this architecture consists of two major parts: the Region Proposal Network (RPN) and the Region of Interest (RoI) Heads. The RPN takes the feature maps from the neck and proposes candidate object regions. Each region is then aligned using RoI Align, and passed through the RoI heads, which contain fully connected layers. These heads are responsible for both classification (predicting the object class) and bounding box regression (refining the coordinates of the predicted boxes). In my implementation, I modified the number of output classes to 11 to align with the task requirements, which include the 10 digit classes (0–9) plus one background class.

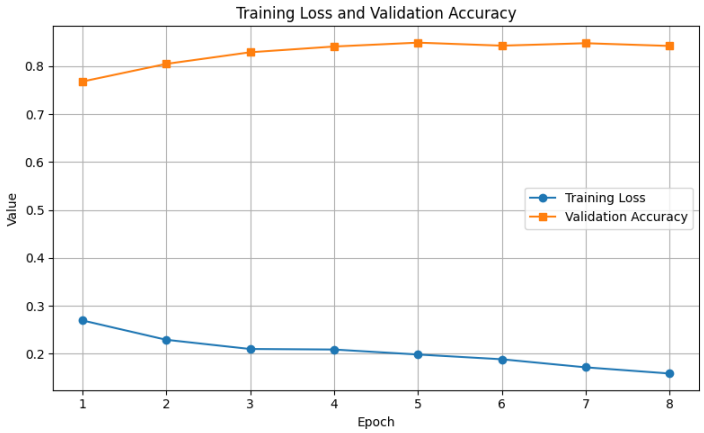
Train strategy:

I observed that the predicted bounding boxes had significant overlap, which negatively impacted the accuracy. To address this issue, I reduced the Non-Maximum Suppression (NMS) threshold from the default value of 0.5 to 0.4, making the suppression stricter and reducing redundant predictions. Additionally, in order to mitigate overfitting and enhance the model's generalizability, I also applied weight decay during training.

Part. 3, Results

(1) Training curve





First, we can see the graph on the left, which shows the training loss and validation mAP over 8 epochs. The blue curve represents the training loss, which shows a steady and consistent decrease, indicating that the model is learning effectively on the training set. In contrast, the orange curve shows the validation mAP. It initially increases, suggesting improved performance on the validation set, and reaches its peak around epoch 5. After that, the mAP begins to decline, which could be a sign of overfitting.

Next, take a look at the graph on the right, which shows the training loss and validation accuracy across 8 epochs. The blue curve represents the training loss, which is the same as the one in the previous graph. The orange curve shows the validation accuracy. It increases consistently during the first 5 epochs, indicating that the model is learning meaningful features and generalizing well. However, after reaching around epoch 5, the curve begins to plateau, showing no improvement in accuracy. This suggests that the model's performance on the validation set has stabilized and further training may not significantly enhance accuracy.

Based on the insights from the two plots, I eventually chose the model weights obtained after fine-tuning for 5 epochs. This checkpoint achieved a mAP of 0.38 on Task 1 and an accuracy of 0.82 on Task 2 when evaluated on the test dataset.

111550061	1	2025-04-14 17:18	266715	111550061	**0.38**	**0.82**

Part. 4, References

Paper: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

GitHub: Weighted-Boxes-Fusion

Part. 5, Additional experiments

After tuning the parameters multiple times, I noticed that my mAP struggled to improve. This led me to hypothesize that a single model might be learning only specific features and may fail to capture certain others, regardless of how much I adjust the parameters or continue training. To address this, I decided to explore whether using multiple Faster R-CNN models with different backbones could enhance the model's ability to localize bounding boxes. In my view, different backbones have varying architectures and levels of complexity, which may allow them to extract different features and finer details from the images.

In my experiment, I utilized five different backbones: ResNet50, ResNet50-FPN v2, ResNet-50 FPN v2 with the first layer frozen, MobileNet, and VGG16. After obtaining the prediction results from each model in the form of pred.json files (containing bounding box information in COCO format), I applied the Weighted Boxes Fusion method to combine their predictions. The Weighted Boxes Fusion I used can be found in the GitHub link listed in the References section. Furthermore, since the ResNet50-FPN v2 and ResNet-50 FPN v2 with the first layer frozen backbone achieved the highest mAP among the models, I assigned them weights of 1.5 in the fusion process, while the other models were each given a weight of 1.0.

As a result, the mAP on the test data increased from 0.38 to 0.41, which can prove my hypothesis is correct. However, despite this improvement, the accuracy in Task 2 dropped significantly. After examining the pred.csv file, I noticed that many predicted results contained one or two non-existent digits. I suspect that this issue arises because some of the weaker models generated bounding boxes in regions where no digits were actually present. Since these incorrect boxes may not be effectively suppressed or merged during the fusion process, they can negatively impact the final digit sequence predictions. Consequently, this reduces the overall accuracy in Task 2.
(At last, I submit the pred.json file form Weighted-Boxes-Fusion method to get mAP 0.41 in task 1 and the pred.csv file from Faster R-CNN with ResNet50-FPN v2 to get accuracy 0.82 in task 2.)

Faster R-CNN with ResNet50-FPN v2:

111550061	1	2025-04-14 17:18	266715	111550061	**0.38**	**0.82**

Weighted Boxes Fusion by using four Faster R-CNN with different backbones:

111550061	1	2025-04-14 18:53	266820	111550061	**0.41**	**0.73**

Part. 6, Code Reliability

(1) Lint the code:

Since I am using a `.ipynb` file, which does not support Flake8 directly, I first convert it to a `.py` file before running Flake8 for code checks.

```
(PRDL) (base) kwchiu@741ge-a6000x4:~/PRDL/HW2$ jupyter nbconvert --to script training.ipynb
[NbConvertApp] Converting notebook training.ipynb to script
[NbConvertApp] Writing 22201 bytes to training.py
(PRDL) (base) kwchiu@741ge-a6000x4:~/PRDL/HW2$ flake8 training.py
(PRDL) (base) kwchiu@741ge-a6000x4:~/PRDL/HW2$ flake8 ensemble.py
(PRDL) (base) kwchiu@741ge-a6000x4:~/PRDL/HW2$
```

(2) GitHub: HW2_LINK