Visual Recognition Using Deep Learning 2025 HW2: Digit Recognition

313551073 顏琦恩

https://github.com/miayan0110/NYCU-Visual-Recognitionusing-Deep-Learning-2025-Spring/tree/master/hw2

I. Introduction

Digit recognition in photographs involves identifying and classifying numerical digits that appear in natural or structured images. Unlike handwritten digit recognition, which deals with relatively clean and centered grayscale digits, this task must address a wider range of challenges, including varying lighting conditions, background clutter, distortions, different font styles, occlusions, and camera angles. The assignment involves several key steps, starting with data preprocessing, which includes loading the image dataset and applying necessary transformations to images. Next, the model is designed using Fast R-CNN as the base model. Finally, I implement the training, validation, and evaluation processes to enable the model to identify digits in real-world photos.

To train the model, I use the dataset provided by the TAs. During training, I validate the model at every epoch to assess the model's performance. Further implementation details will be discussed in the next section.

II. Method

I implemented the digit recognition model using Fast R-CNN architecture with a ResNet-50 backbone, a Feature Pyramid Network (FPN) neck, and the head consists of the Region Proposal Network (RPN) and the Region of Interest (RoI) heads, realized via the fasterrcnn_resnet50_fpn_v2 model from torchvision. Since the digit cannot be flipped, I only applied normalization to the images to help stabilize the training process.

The backbone ResNet-50 is pretrained on ImageNet. It is composed of an initial convolution and pooling layer followed by four residual blocks. In my setting, only three stages of backbone are set as trainable to allow fine-tuning on our specific dataset, enabling the model to better adapt to the visual characteristics of photographic digits.

The Feature Pyramid Network (FPN), which acts as the neck of the model. The FPN enhances multi-scale feature representation by constructing a top-down pyramid structure with lateral connections, allowing the detector to be sensitive to objects of various sizes and resolutions.

To adapt the model for digit detection, I replaced the original classification head with a new FastRCNNPredictor, setting the number of output classes to 11 (digits 0–9 plus background).

The model is trained for 10 epochs with a batch size of 8, using Stochastic Gradient Descent (SGD) with a learning rate of 0.005, momentum of 0.9, and weight decay of 1e-4.

```
def initialize_model(resume=False, device='cuda'):
    logdir = './records'
    os.makedirs(logdir, exist_ok=True)
    writer = SummaryWriter(logdir)

    model = torchvision.models.detection.fasterrcnn_resnet50_fpn_v2(
        weights=FasterRCNN_ResNet50_FPN_V2_Weights.DEFAULT,
        trainable_backbone_layers=3
    )
    num_classes = 11
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
    model = model.to(device)

    optimizer = torch.optim.SGD(
        model.parameters(),
        lr=0.005,
        momentum=0.9,
        weight_decay=1e-4
    )
    start_epoch = 0

    if resume:
        ckpt_path = sorted(glob.glob('./ckpt/trainable_3/*.pth'))[-1]
        model, optimizer, start_epoch = load_checkpoint(
            model,
            optimizer,
            ckpt_path,
            device=device
        )

    return model, optimizer, start_epoch, writer
```

III. Results

During training, I saved a checkpoint at the end of each epoch and kept track of the training loss to generate a loss curve (Fig. 1). I also computed the mean Average Precision (mAP) at each epoch to evaluate the model's detection performance (Fig. 2). As shown in Fig. 1, the loss steadily decreases over 10 epochs, indicating that the model is learning effectively. However, from Fig. 2, we can see that the mAP curve fluctuates across epochs—initially showing an improvement, but not monotonically increasing throughout the training. This behavior may suggest that hyperparameter settings, data variance, or partial overfitting could be influencing the final results.

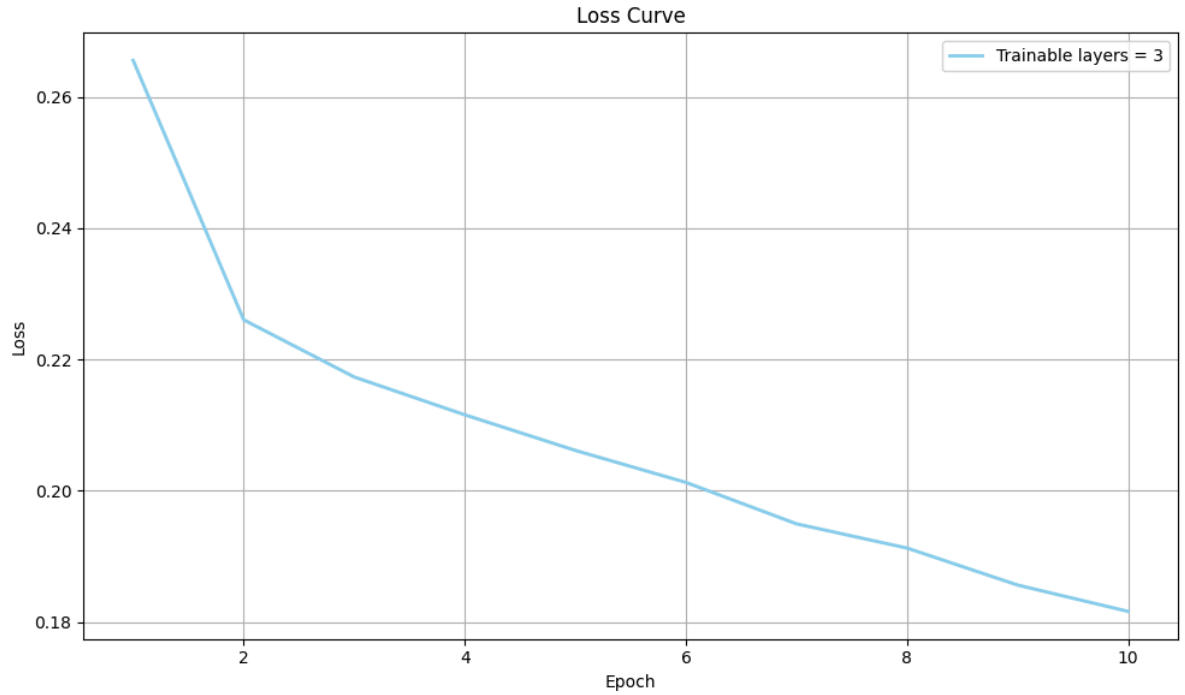

Figure 1. Loss curve.

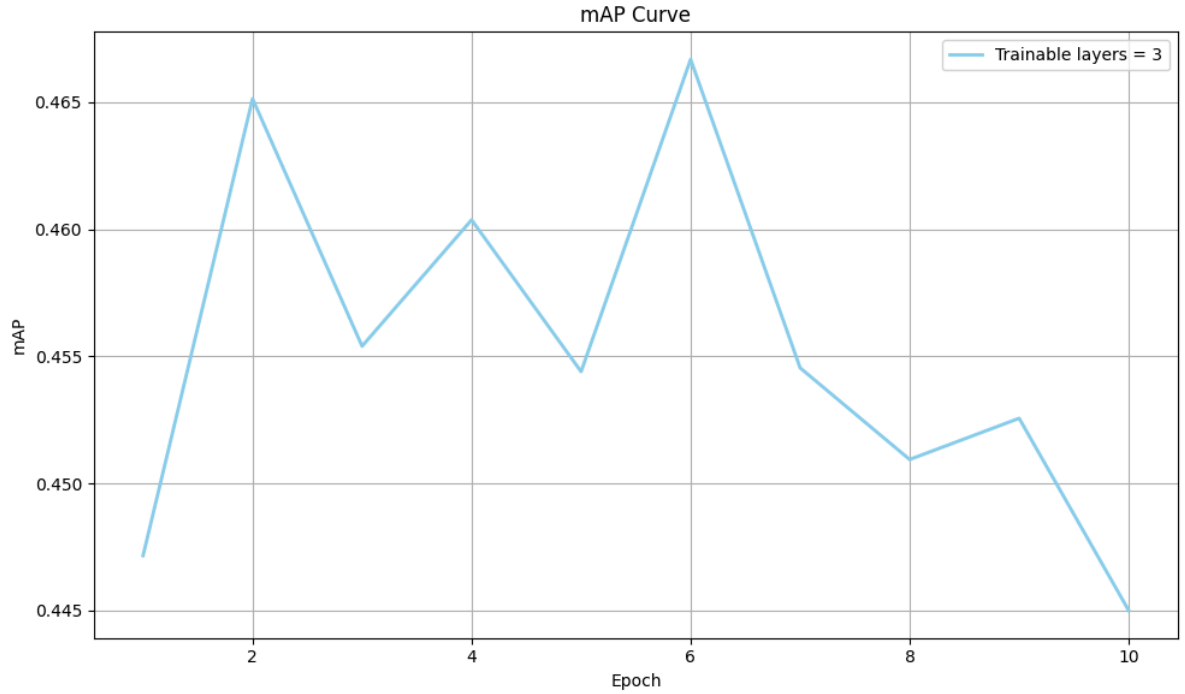

Figure 2. mAP curve.

IV. References

1. Mean Average Precision (mAP)
   - https://lightning.ai/docs/torchmetrics/stable/detection/mean_average_precision.html
   - https://stackoverflow.com/questions/78359307/how-to-use-meanaverageprecision-metric-from-torchmitrics-detection-on-an-object

V. Additional Experiments

I conducted two other settings to the model for additional experiments. The first one is to change the number of training layers. In the experiment, the trainable layer number is set to two, while the original setting is three. Another experiment is about data augmentation, which involves training models with data without normalization.

From Fig. 3, we can see that the loss curves of these modifications are similar to the original loss curve, indicating that the overall convergence behavior, in terms of loss reduction, is not severely impacted by these changes. However, the mAP score from Fig. 4 shows that the original setting achieves the best detection performance. This suggests that while the adjustments in the number of trainable layers and data normalization have a limited effect on the training loss, they can significantly influence the final detection quality.

In particular, having three trainable layers with proper normalization appears to strike a better balance between retaining pretrained features and adapting to the specific dataset, thereby yielding higher mAP scores. These experiments highlight that even small modifications in model configuration and data preprocessing can lead to notable differences in detection performance.

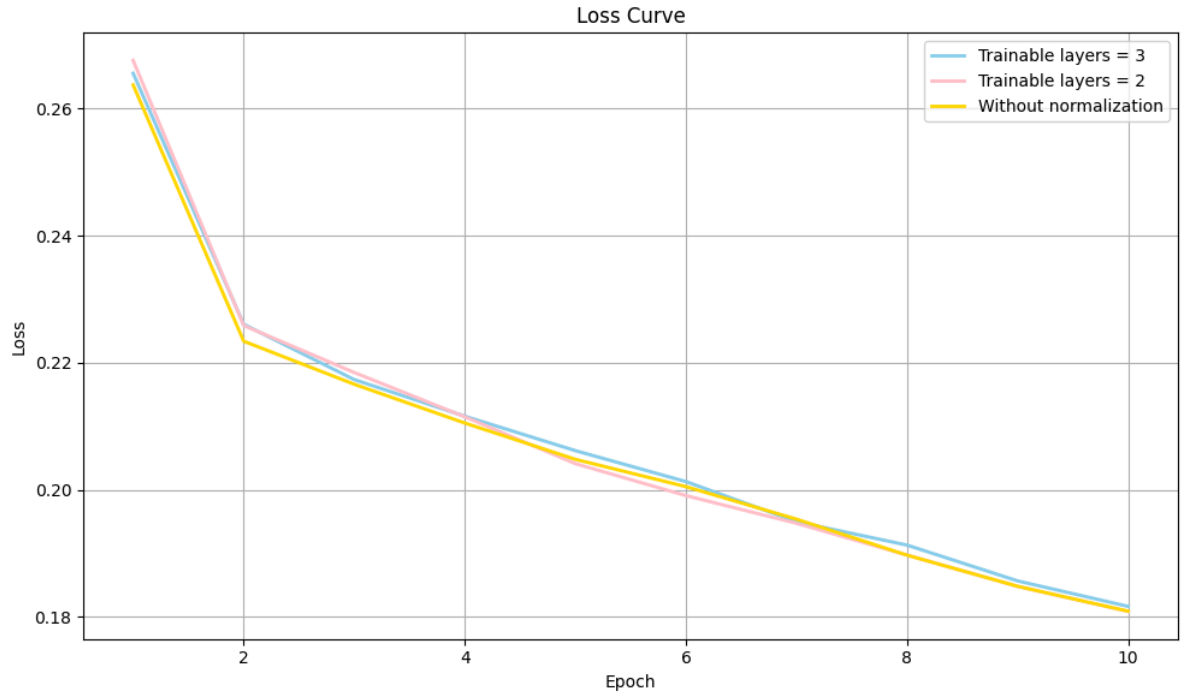

Figure 3. Loss curve of experiment.

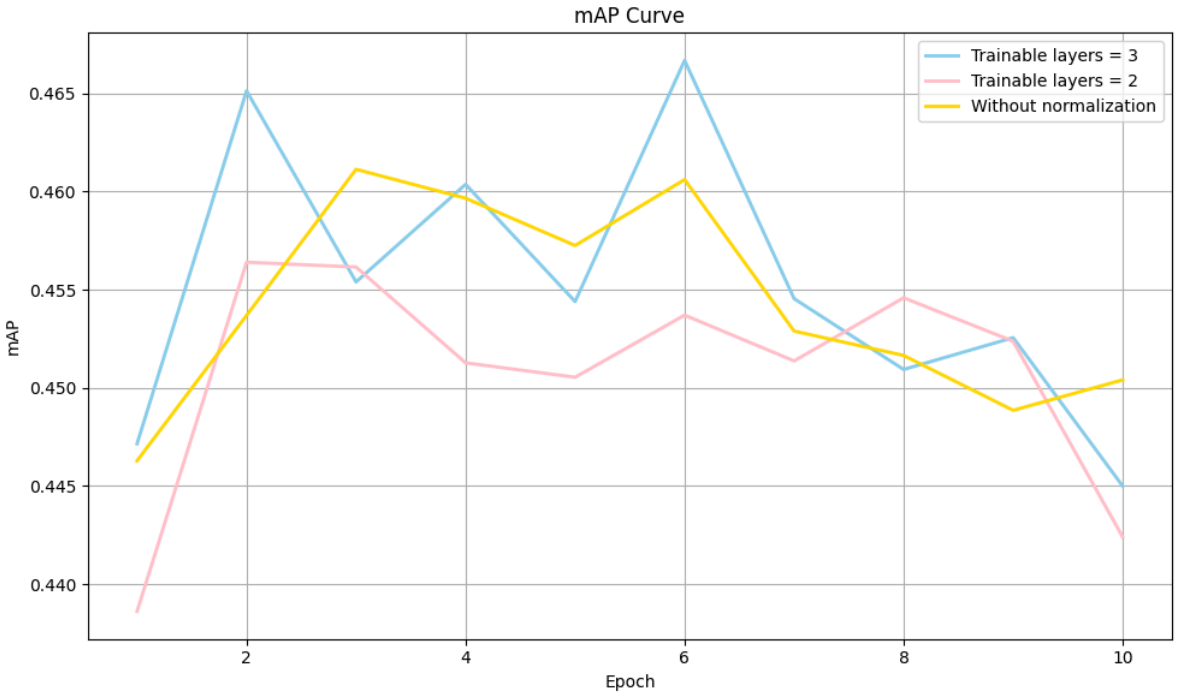

Figure 4. mAP curve of experiment.