Visual Recognitionusing Deep Learning

2025 Spring, Homework 2

313551078 吳年茵

Github : https://github.com/nianyinwu/CV_HW2

1. Introduction

In this lab, we have to train a Faster R-CNN [1] as our model to solve a digit recognition task with 11 classes (including background) and design other tricks to improve the model's performance, such as modifying the architecture. This lab has two tasks: task 1 is needed to recognize the class and bounding box of each digit in the image; task 2 is for the number of detected digits in the image. We use a dataset of 30,062 RGB images for training, 3,034 for validation, and 13,068 for testing.

To improve the model's performance for two tasks, I adopt an improved Faster R-CNN model with a ResNet-50-FPN backbone constructed by this paper [2], named Faster R-CNN v2 in pytorch [3], which modified the backbone and detection module to have better metrics than Faster R-CNN. I also use pre-trained weights from training on COCO_V1 to converge better and learn current training data based on previously learned visual features. Additionally, I experiment with some tricks, such as making some data augmentations or introducing the Convolutional Block Attention Module (CBAM) [4], which sequentially applies channel attention followed by spatial attention that is intended to help the model emphasize informative features across both dimensions and enhance its ability to focus on meaningful regions within the input.

2. Method

A. Data Preprocessing

I do some data preprocessing and augmentation techniques to the data using PyTorch's transforms module [5].

The preprocessing process of training data is as follows:

1. Randomly flip the image horizontally with a probability of 0.5.
2. Apply color jitter with brightness ±20%, contrast ±20%, and saturation ±10%.
3. Randomly adjust the image sharpness with a factor of 10 and a probability of 0.8.
4. Convert the image into a PyTorch tensor.
5. Convert the tensor to float32 and scale pixel values from [0, 255] to [0.0, 1.0].

The preprocessing process of validation and testing data is as follows:

1. Convert the image to a PyTorch tensor.
2. Convert the tensor to float32 and scale pixel values from [0, 255] to [0.0, 1.0].

B. Model Architecture

B.1. Base Architecture

1. Backbone

   The backbone in the Faster R-CNN v2 used in this work is ResNet-50 [6], the same as in the original Faster R-CNN. It extracts hierarchical and discriminative visual features that guide the model in identifying important regions within the input image. The main difference between the two models is the batch normalization layers. In the original Faster R-CNN, the batch normalization layers are frozen and do not update during training, whereas in Faster R-CNN v2, they are trainable. Freezing these layers may weaken the model's ability to adapt to unseen data. Due to hardware limitations, a deeper backbone such as ResNet-101 was not adopted. Though it may cause higher performance, it would significantly increase the training time. Therefore, the backbone architecture was kept unchanged.

2. Neck

   After the backbone extracts features, the model employs a Feature Pyramid Network (FPN) to fuse multi-scale features from different backbone layers. This allows the model to effectively handle objects of various sizes by leveraging features at

multiple resolutions and, in this part, using an extra batch normalization layer after each convolutional layer.

3. Head

   Subsequently, the model employs a Region Proposal Network (RPN) to generate a set of object proposals by sliding a small network over the feature maps. For each anchor, it predicts an objectness score and refines the bounding box coordinates. After applying non-maximum suppression (NMS), the top-N proposals are forwarded to the RoI head for further classification and regression. In this stage, compared to the original Faster R-CNN, the v2 version adopts two convolutional layers in the RPN module instead of one, potentially improving its capacity to generate more accurate proposals.

   Finally, the Region of Interest (RoI) head takes the top-N proposals from the RPN and extracts fixed-size feature maps for each proposal using RoI Align. Then, the feature passed through a series of fully connected layers to perform classification and bounding box regression. Specifically, the RoI head predicts the class label for each proposal and further refines its coordinates to fit the target object better. This stage enables the model to localize and accurately multiple objects within the input image. In this part, the v2 version uses four convolutional layers with batch normalization followed by a linear layer as a heavier box regression head compared to the original.

B.2. Modification (including additional experiments modification)

In this section, I will present the modification in the base architecture and the modification of additional experiments. In all experiments, I modified the number of classes in the ROI head that predict classes from 91 to 11 (background plus 10-digit classes) to match the task dataset.

When observing the dataset image, I found that most digits are small and often surrounded by background noise. Thus, in additional experiments, I wanted to add a Convolutional Block Attention Module (CBAM) [4] into every layer of the backbone network except the first layer. Since CBAM sequentially applies channel and spatial attention, it may help the model learn more informative and fine-grained features along both dimensions, enhancing its ability to focus on meaningful regions. So, I want to solve this problem and improve model performance by introducing the attention mechanism.

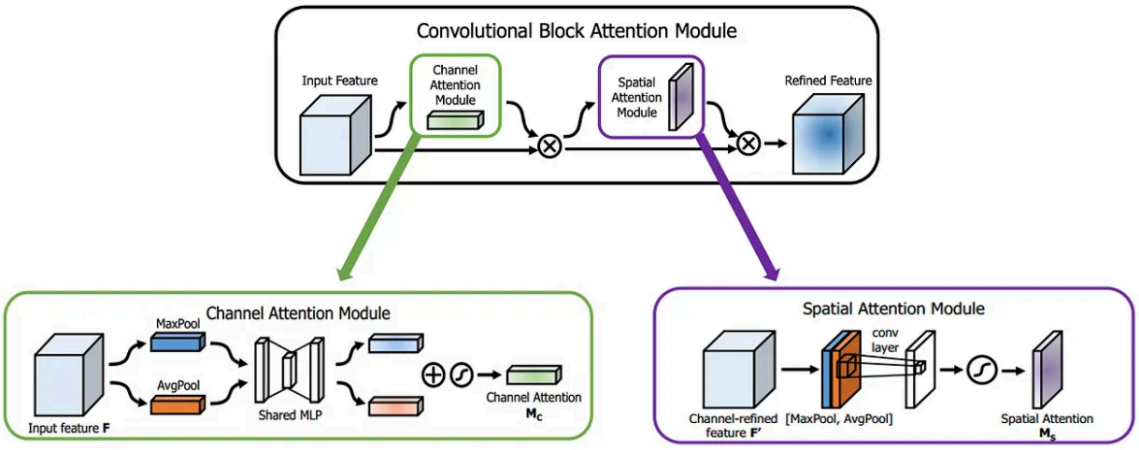


Fig. 1. CBAM module from [7]

C. Hyperparameters Setting

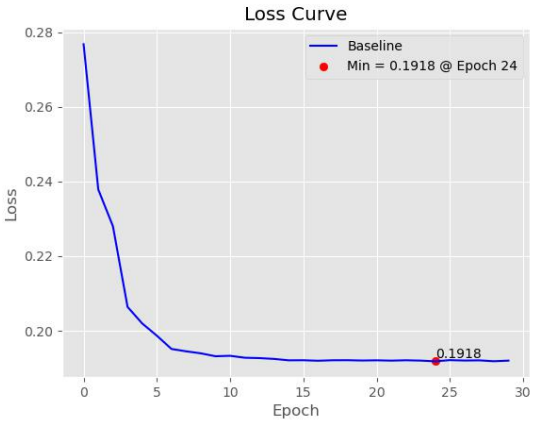
In the experiments, I use the following hyperparameter settings to train the model:

- Learning rate: 5e-3
- Batch size: 4
- Number of epochs: 30
- Optimizer: SGD with momentum=0.9 and weight_decay=5e-4
- Learning rate scheduler: Cosine annealing with linear warmup during the first 5% of the total training steps, implemented using Hugging Face's library [8] .

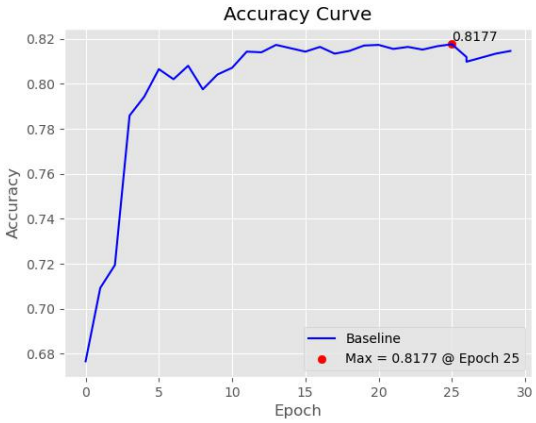
3. Experiment Results

I conducted three experiments to explore better performance: (1) Faster R-CNN v2 only modified the number of classes in the ROI head, and using a StepLR learning rate scheduler. (baseline) (2) Added CBAM module into every layer of the Faster R-CNN v2 backbone except the first layer and also using StepLR. (3) Adopt the same architecture as (2), but using the cosine annealing scheduler with first 5% of total training epochs linear warmup instead StepLR.

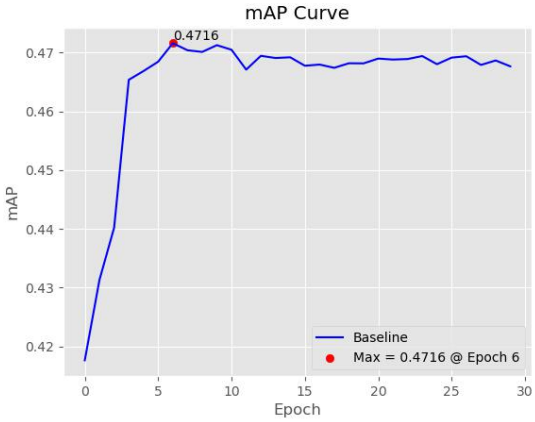
1. Faster R-CNN v2 (baseline)



(a) Loss Curve



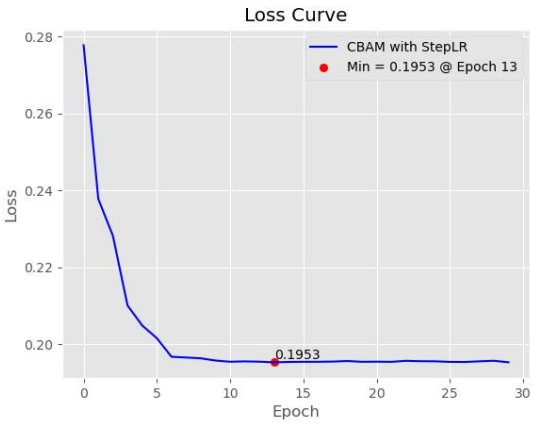
(b) Accuracy Curve



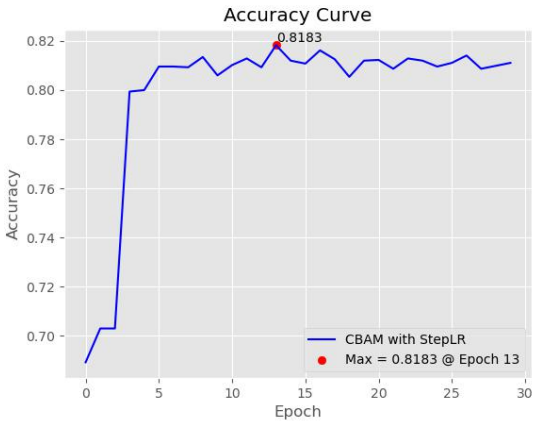
(c) mAP Curve

In the first experiment, Faster R-CNN (baseline) only modified the number of classes in the ROI head from 91 to 11. The loss curve shows that the model converges quickly after epoch 10. The lowest loss reached is **0.1918**, the highest validation accuracy achieved is **0.8177**, and the best mAP is **0.4716** at epoch 6, with limited improvement afterward. Thus, I thought this setting's generalization ability reached a plateau early, and performance was not very well that I expected.

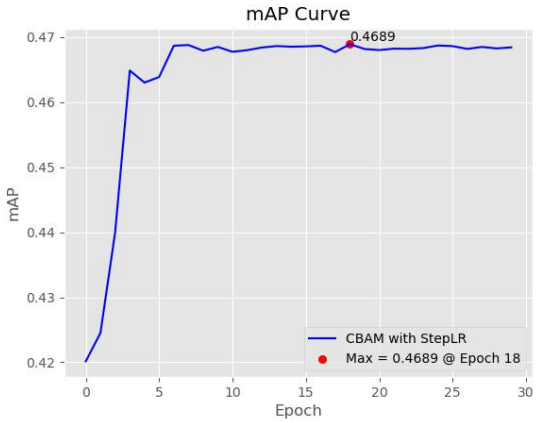
2. Faster R-CNN v2 + CBAM module



(a) Loss Curve



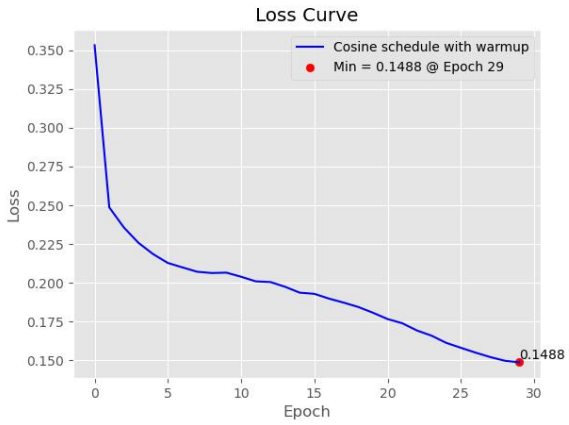
(b) Accuracy Curve



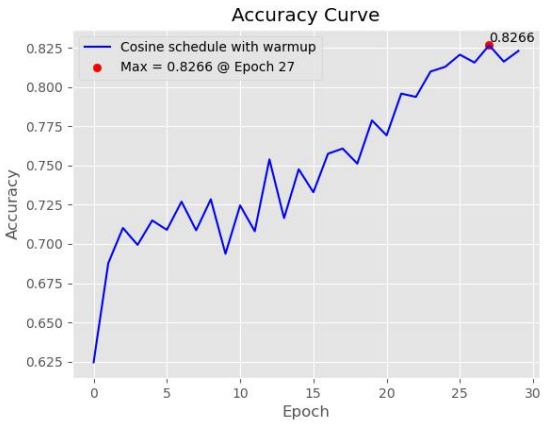
(c) mAP Curve

As a result of the second experiment that only added the CBAM module into every layer of the Faster R-CNN v2 backbone except the first layer and also using StepLR, the validation accuracy **0.8183** but its loss not lower than the first experiment, and the highest mAP is **0.4689**, comparable to the baseline. So, although the addition of CBAM appears to enhance feature refinement (as evidenced by its slightly higher validation accuracy compared to the baseline), it does not significantly improve accuracy or mAP under the StepLR schedule. In fact, the mAP performance is slightly worse. I guess that the benefit of CBAM may be limited by the fixed-step learning rate decay, as reflected by the overall higher loss compared to the baseline. Thus, I try another learning rate scheduler in the third experiment.

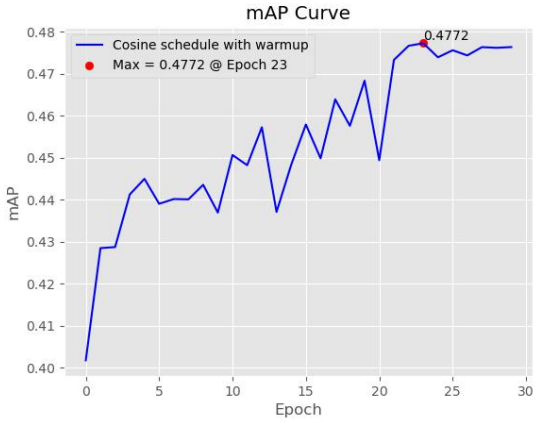
3. Faster R-CNN v2 + CBAM module with cosine annealing scheduler



(a) Loss Curve



(b) Accuracy Curve



(c) mAP Curve

In the third experiment, I retained the CBAM modules and replaced the StepLR scheduler with a cosine annealing learning rate schedule, incorporating a 5% linear warmup. As shown in the loss curve, the model achieved the lowest loss (**0.1488**) among all experiments. Additionally, the validation accuracy surpassed **82%**, and the mAP reached the highest value across the three settings. While the validation accuracy and mAP curves exhibit more noticeable fluctuations than the other settings, this is likely due to the nature of cosine annealing, which introduces more considerable learning rate variations—especially after the warmup phase. Despite this, both accuracy and mAP show a consistent upward trend without early saturation, suggesting that cosine annealing with warmup contributes to more effective training dynamics.

And I thought that extending the number of training epochs could further improve performance in this setting. However, only 30 training epochs were used due to the training time. And in the codabench, I selected this model inference result as my final submission, and during training, I saved the weights of the highest validation accuracy.

References

[1] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[2] Yanghao Li, Saining Xie, Xinlei Chen, Piotr Dollar, Kaiming He, and Ross Girshick. Benchmarking detection transfer learning with vision transformers, 2021.

[3] https://pytorch.org/vision/main/models/faster_rcnn.html.

[4] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module, 2018.

[5] http://pytorch.org/vision/main/transforms.html.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[7] https://medium.com/@andy6804tw/通道空間注意力-cbam-805deb9ebe1c.

[8] https://huggingface.co/docs/transformers/main$_c lasses/optimizer_s chedules$.