NYCU Visual Recognition using Deep Learning HW2 report

109612019, 林伯偉

Github Link : https://github.com/Albert5865/NYCU-Computer-Vision-2025-Spring/tree/main/HW2

1. Introduction:

In this assignment, I applied **Faster R-CNN** with a **ResNet-50 backbone** to detect and recognize digits within images. The task involved detecting the class and bounding box for each digit, and the overall goal was to evaluate the performance using **mean Average Precision (mAP)**. To enhance model performance during training, I experimented with different learning rate schedulers, particularly focusing on the **CosineAnnealingLR** scheduler, which adjusts the learning rate in a smooth cosine curve, making it ideal for the model to converge gradually.This report outlines the method used to train Faster R-CNN, highlights the effectiveness of **CosineAnnealingLR** as the learning rate scheduler, and discusses the results and experiments conducted.

2. Method:

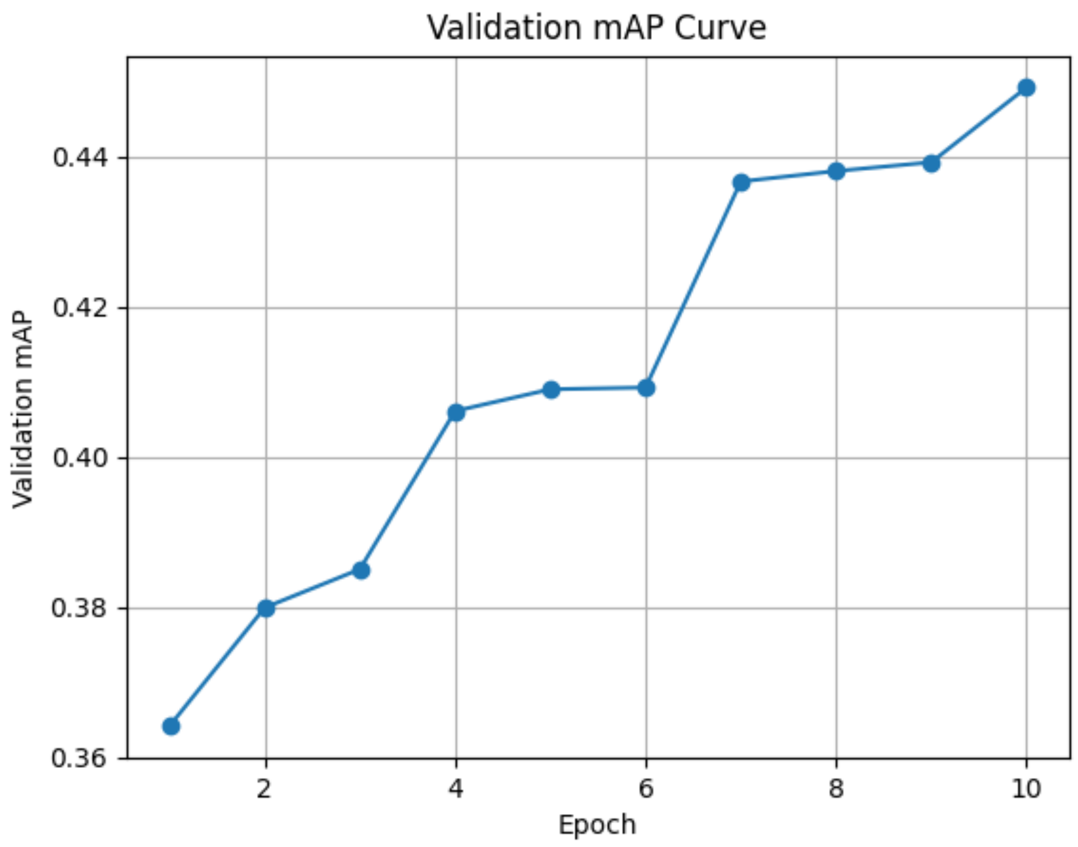
1. **Model Architecture :** Used **Faster R-CNN-fpn-V2**, a widely used object detection model that integrates a Region Proposal Network (RPN)with a deep backbone (ResNet-50). The model has three key components:
2. **Region Proposal Network (RPN)** : This module proposes potential bounding boxes that may contain objects of interest.
3. **Head** : The head of the network predicts the class of each object and refines the bounding boxes.

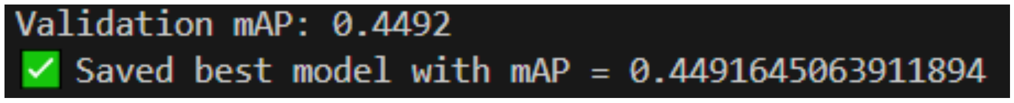
Faster R-CNN was implemented using the **torchvision** package in PyTorch, with the **ResNet-50** backbone pre-trained on COCO. The task required detecting the digits present in the images, predicting the bounding boxes, and classifying each detected digit. The performance of the model was evaluated using **mAP**. For each epoch, **10%** of the training data was randomly selected to reduce memory usage, which allowed the model to handle large datasets without running into memory constraints.

To improve training efficiency, I used the **CosineAnnealingLR** scheduler, which gradually reduces the learning rate following a cosine curve. The cosine annealing function ensures that the learning rate decreases smoothly, promoting better convergence as the training progresses.

- Model : fasterrcnn_resnet50_fpn_v2
- Learning rate : starts at 1e-4
- Weight decay : 5e-4
- Epoch : 10
- Optimizer : AdamW

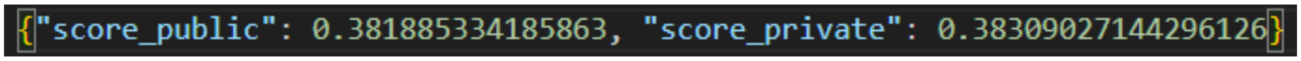
3. Results:





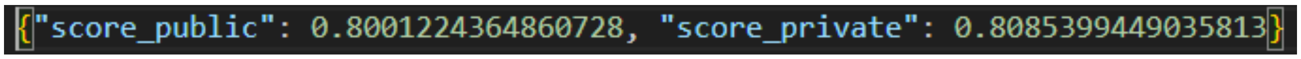
```
Validation mAP: 0.4492
✅ Saved best model with mAP = 0.4491645063911894
```

- Task 1 score :



```
{"score_public": 0.381885334185863, "score_private": 0.38309027144296126}
```

- Task 2 score :



```
{"score_public": 0.8001224364860728, "score_private": 0.8085399449035813}
```

The model was trained for **10 epochs**, with **10% of the training data** randomly sampled for each epoch. The results showed a steady improvement in the model’s performance.The validation mAP improved as the training progressed, showing that the model generalized well to unseen data.At the end of 10 epochs, the best validation mAP achieved was **0.449**, demonstrating the model's ability to accurately detect and classify digits.

4. Additional experiments:

- **Different Scheduler :**

  - **Hypothesis**: I hypothesized that different learning rate schedulers would impact the model's ability to converge during training. Specifically, experimented with the **StepLR** scheduler, which decreases the learning rate by a fixed factor every few epochs. Expected that this type of scheduler would allow the model to fine-tune after an initial rapid descent in the learning rate, leading to better generalization.

  - **How it works**: The **StepLR** scheduler decays the learning rate at fixed intervals (step_size), by a factor of gamma. The formula is as follows:

$$\eta t = \eta t - 1 \times \gamma \text{ if epoch mod step\_size} == 0$$

  Where:
  - ηt is the learning rate at current epoch
  - $\eta t - 1$ is the learning rate at the previous epoch.
  - γ is the factor by which the learning rate will be reduced (0.7 in this experiment).

- **Random Sampling to reduce memory use:**

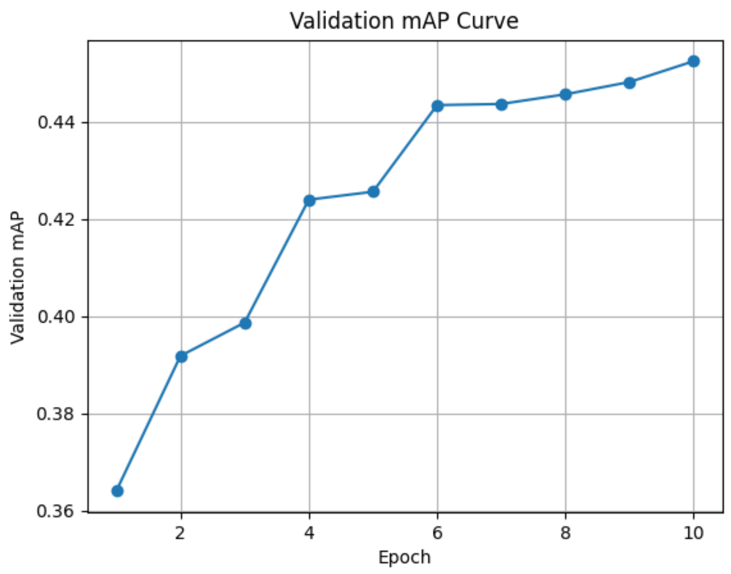
  - **Hypothesis**: I hypothesized that **random sampling** of a smaller subset of the training data in each epoch would help reduce memory usage during training. Specifically, by using only **10% of the dataset** randomly each epoch, expected to be able to train on larger datasets that would otherwise not fit into memory (only 16GB on my device). This technique should allow the model to learn effectively **without freezing.**

  - **How it works**: In each epoch, instead of using the entire training dataset, we **randomly selected 10%** of the dataset. This subset was then used to train the model for that epoch. The random selection was done for each epoch, ensuring the model saw different data points throughout the training process

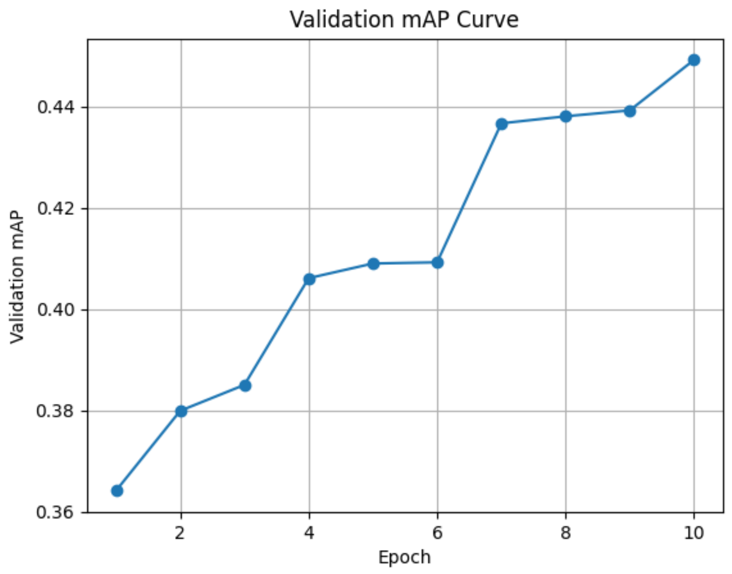
Despite training on a smaller subset each time, the model was still able to generalize well(this method is utilized through out the whole assignment), and the performance on the validation set improved steadily. This approach helped me manage memory constraints and prevent my device from freezing.

Additional experiment results:

StepLR Scheduler



CosineAnnealingLR Scheduler



The **first curve** shows the **mAP curve** for the model using the **StepLR** learning rate scheduler. With **StepLR**, the learning rate decays at fixed intervals (every epochs) by a constant factor, which causes the mAP to exhibit a **stepped increase**during training. The mAP gradually increases with each epoch, but the curve shows noticeable "jumps" at each learning rate step. While the model does show improvement over time, the progression is not as smooth as in the **CosineAnnealingLR** curve, and the improvements are not as consistent between the epochs.

The **second plot** represents the **mAP curve** for the model using **CosineAnnealingLR**. This scheduler reduces the learning rate smoothly following a cosine curve. The resulting mAP curve is more **smooth and gradual**, showing a steady increase without abrupt jumps. The mAP steadily improves across epochs, with a more continuous progression in comparison to the **StepLR** model. The smoothness of the curve indicates that the model benefited from a more gradual decrease in learning rate, which likely achive a higher mAP if the model was trained for a even higher count of epoch.

Comparison

1. **Smoothness of Progression**:
    - The **StepLR** scheduler leads to abrupt changes in mAP as the learning rate decreases in steps.
    - The **CosineAnnealingLR** scheduler provides a smoother curve, indicating that the model is able to converge more gradually and steadily.

2. **Convergence Rate**:
   - With **StepLR**, the model reaches a decent mAP, but the growth is slower and less continuous.
   - With **CosineAnnealingLR**, the model's mAP increases more consistently, with smoother and more uniform progress.

3. **Effectiveness**:
   - While both schedulers improve mAP over time, the **CosineAnnealingLR** scheduler seems to provide better and more stable results throughout the training process. The smooth decay of the learning rate likely achive a higher mAP if the model was trained for a even higher count of epoch.

References:

1. **Faster R-CNN**: Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015.
2. **CosineAnnealingLR Scheduler**: PyTorch documentation on learning rate schedulers. (https://pytorch.org/docs/stable/optim.html#cosineannealinglr)
3. **PyTorch Documentation**: Official documentation for Faster R-CNN in torchvision (https://pytorch.org/vision/stable/models.html#faster-r-cnn)
4. **COCO Evaluation**: PyCocoTools for object detection evaluation. (https://github.com/cocodataset/cocoapi)