NYCU Visual Recognition using Deep Learning Homework 2

[312551185], [郭晏涵]

GitHub Link:

https://github.com/slovengel/312551185_HW2

Introduction:

This project aims to tackle a digit recognition problem using Faster R-CNN. The dataset consists of RGB images, with 30,062 samples for training, 3340 samples for validation, and 13068 samples for testing. For task 1, the goal is to recognize the class and bounding box of each digit in the image. For task 2, the goal is to predict the whole number in the image using the results from Task 1.

To enhance digit recognition performance, I employed several key strategies including transfer learning, adaptive learning rate scheduling, and post-processing techniques. The core of my method is built upon a Faster R-CNN framework with a ResNet-50-FPN backbone, fine-tuned for digit detection. MeanShift clustering filters out outliers, and digits are ordered spatially to reconstruct full numbers.

The proposed model achieved 0.39 mAP on task 1 and 82% accuracy on task 2 in the CodaBench competition (on public leaderboard), demonstrating its effectiveness in handling the recognition task.

Method:

1. **Data pre-processing**

   The entire dataset images are converted to tensors and normalized using ImageNet mean and standard deviation. The original train/validation split was used. Unlike traditional classification pipelines, the images are not resized to a fixed size, as object detection tasks require preserving spatial structure and scale information.

   Initially, I experimented with common augmentation techniques like RandomHorizontalFlip, RandomRotation and ColorJitter to improve generalization and reduce overfitting. However, these transformations led to a drop in both training and validation mAP, so I chose not to apply them in the final pipeline. This is likely because such augmentations distort digit shapes or orientations in ways that confuse the detector.

2. **Model architecture**

   The model is built upon the Faster R-CNN framework, which consists of three main components: a backbone, a neck (RPN), and a head.

- Backbone: We use a ResNet-50 with Feature Pyramid Network (FPN) as the feature extractor. The ResNet-50 backbone captures rich spatial features at multiple scales, while the FPN enhances performance on small objects like digits by combining low and high-resolution features.
- Neck (RPN): The Region Proposal Network generates bounding boxes that are likely to contain digits. It operates on multi-scale feature maps from the FPN and proposes ROI based on anchor boxes and objectness scores.
- Head: The detection head consists of two branches:
  - A classification branch that predicts digit class scores.
  - A regression branch that refines the bounding box coordinates.

The default prediction head is replaced with a new one customized for digit classification by setting the number of output classes to 11 (10 digits + background).

3. Hyperparameters

The three models are trained using **SGD** with momentum of 0.9 and a **batch size** of **4** for 10 epochs. **Weight_decay** of **1e-4** is used of prevent overfitting.

CosineAnnealingLR scheduler is applied to adjust the learning rate dynamically throughout training. It creates a smooth and continuous decay of the learning rate without interruptions. The learning rate starts at **1e-3** in the first epoch and gradually decreases to **1e-6** by the final epoch.

4. Whole Number Recognition

To recognize whole numbers from digit-level detections, we apply a post-processing pipeline that reduces noise and enhances accuracy. First, low-confidence and overlapping detections are filtered using a confidence threshold of 0.4 and Non-Maximum Suppression (NMS). To eliminate outliers, the remaining detections are clustered based on their spatial centers using the MeanShift algorithm. For special cases with exactly three detections, a custom rule filters improbable outliers based on center spacing ratios. After filtering, the digits are sorted by horizontal position and concatenated to form the final number prediction.

Additional experiments:

1. Backbone Comparison

This experiment investigates the impact of different backbones on the performance of the Faster R-CNN framework for digit recognition. Specifically, we compare fasterrcnn_resnet50_fpn_v2 and fasterrcnn_mobilenet_v3_large_320_fpn.

The detailed implementation is as follows:

1. Train two Faster R-CNN models independently using the same training pipeline, differing only in the backbone architecture.

2. The fasterrcnn_resnet50_fpn_v2 model uses a ResNet-50-FPN backbone derived from the Benchmarking Detection Transfer Learning with Vision Transformers [1] paper.
3. The fasterrcnn_mobilenet_v3_large_320_fpn model is a lightweight variant optimized for low-resolution input (320×320) and fast inference.

The following are the experiment results:

Model Name	Best Validation mAP
fasterrcnn_mobilenet_v3_large_320_fpn	0.35
fasterrcnn_resnet50_fpn_v2	0.39

Analysis:

The ResNet-50-FPN backbone outperforms the MobileNetV3-Large backbone due to its deeper architecture and strong feature extraction capabilities. Its residual connections and multi-scale feature aggregation via the Feature Pyramid Network (FPN) allow it to capture both spatial and semantic information effectively. This is especially advantageous for detecting small and densely packed digits, as it enables more accurate localization and classification.

On the other hand, the MobileNetV3-Large backbone is optimized for speed and efficiency, but its shallower depth limits its ability to model complex patterns. Combined with its lower input resolution, the model struggles with identifying small or closely spaced digits, leading to reduced detection performance.

2. Clustering Algorithm Comparison

This experiment compares two clustering-based filtering methods—MeanShift and DBSCAN—used to remove outlier detections during whole number recognition. Both algorithms cluster detected digit bounding boxes based on their center coordinates and retained only the digits within the largest cluster.

Implementation differences:

- MeanShift estimates the number of clusters automatically using a quantile-based bandwidth (set to 0.4). A default bandwidth of 30 is used when estimation fails, allowing adaptive clustering across varied layouts.
- DBSCAN requires manually defined parameters: eps=30 (maximum neighbor distance) and min_samples=1 (minimum points to form a cluster). These fixed values can make it less flexible when detection spacing varies.

The following are the experiment results:

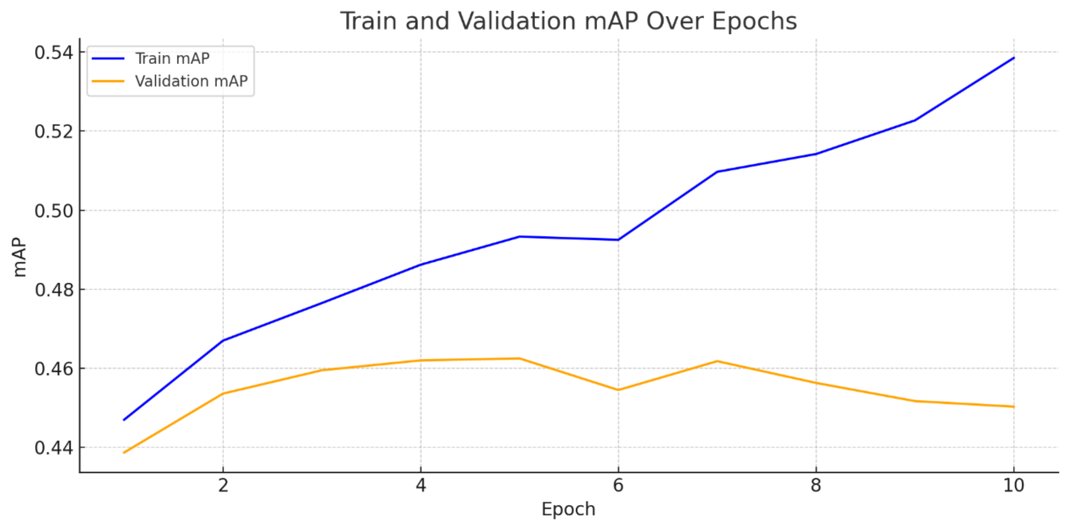
Model Name	**Best Validation Acc.**
DBSCAN	0.75
MeanShift	0.82

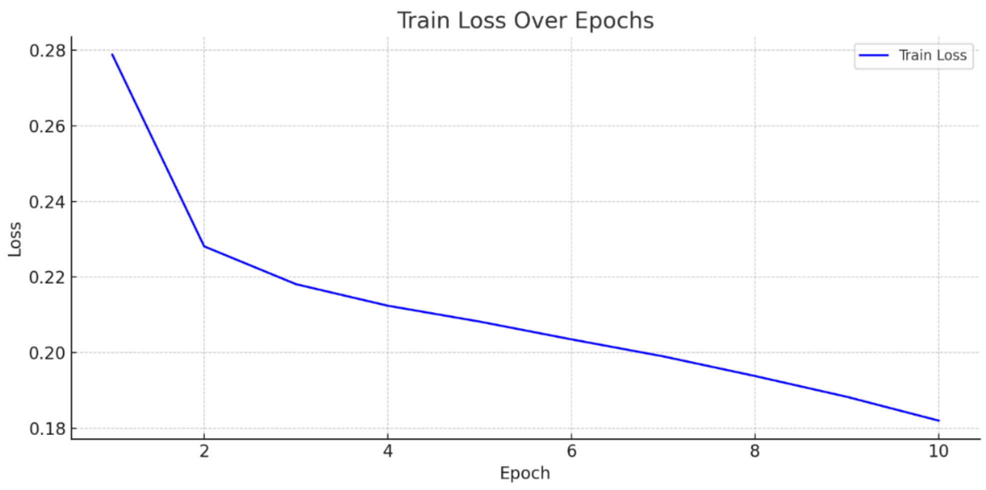
Analysis:

MeanShift outperforms DBSCAN due to its robustness in handling small numbers of detections. DBSCAN depends heavily on parameters like eps and min_samples, which are difficult to tune when data points are few and unevenly spaced. It may misclassify valid digits as outliers or form incorrect clusters. On the other hand, MeanShift adaptively estimates bandwidth and is less sensitive to point density, making it more effective in isolating the main digit group and removing spurious detections. This adaptability gives MeanShift a consistent advantage in low-count scenarios.

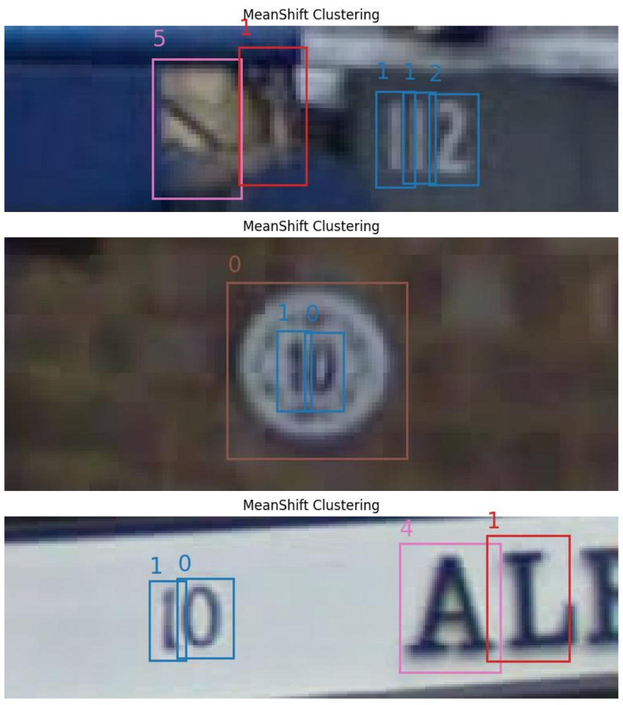
Results:

Based on the experimental results, neither lighter backbones nor DBSCAN provided significant improvements. Since the combination of the fasterrcnn_resnet50_fpn_v2 backbone and MeanShift clustering achieved the best performance, it was selected as the final model for inference. The following are the learning curves for training the proposed model.





Early stopping was applied at epoch 5 to mitigate potential overfitting, as the validation mAP began to plateau and slightly decline. This strategy helps preserve the model's generalization ability and avoids fitting noise in the training data.



The figure above shows examples of whole number recognition using MeanShift clustering. The blue bounding boxes represent digit detections that belong to the largest cluster, which is assumed to form the target whole number. The non-blue bounding boxes indicate noise or non-target digits that were filtered out during the clustering process.

Code Reliability:

I lint the code using Flake8 to follow the PEP8 instructions.

References:

[1] Benchmarking Detection Transfer Learning with Vision Transformers https://arxiv.org/abs/2111.11429