[Github link] [Experiment Result]

Introduction

Task Overview

The objective of this project is to perform digit recognition using the Faster R-CNN framework. The task is divided into two parts. Task 1 focuses on detecting individual digits within an image, where the model must predict bounding boxes and classify each digit using COCO-format annotations. Each predicted object must include the image ID, bounding box coordinates (unnormalized), a confidence score, and a category ID starting from 1 (where category ID 1 corresponds to the digit "0"). Task 2 involves recognizing the complete digit sequence from the image (e.g., "49", "112") and submitting the results in a CSV file with two columns: image_id and pred_label. Only Faster R-CNN is allowed, but modifications to its backbone, neck (Region Proposal Network), and head are encouraged and may lead to higher report scores.

The dataset consists of RGB images with labeled digits provided in COCO format. It includes 30,062 images for training, 3,340 for validation, and 13,068 for testing. Bounding boxes are defined in the format [x_min, y_min, width, height], and no normalization is applied.

Method Overview

In this work, I employed the PyTorch implementation of Faster R-CNN as the backbone model and trained it on an augmented version of the dataset. The focus was placed on applying stronger data augmentation techniques to improve the model's generalization and overall performance. During training, I evaluated the model using the mean Average Precision (mAP) metric, which provides a comprehensive measure of detection accuracy. For inference, a confidence score threshold was introduced as a hyperparameter to filter low-confidence predictions, thereby enhancing the quality of the final results.

Data Preprocessing

Data Augmentation

DigitCocoDataset

Methods	Discriptions
`Resize((256, 256))`	• Fix training data size.
`T.ColorJitter(brightness=0.4, contrast=0.4,`	• Brightness/Contrast/Saturation: Adjusted within

`saturation=0.4, hue=0.3)`	**±20%**. • Hue: Adjusted within **±10%**, to retain correct color feature.
`T.RandomGrayscale(p=0.1)`	• Converts the image to grayscale with a 10% probability.
`T.RandomAdjustSharpness(sharpness_factor=2, p=0.3)`	• Randomly sharpens the image (factor=2) with a 30% probability.
`T.GaussianBlur(kernel_size=3, sigma=(0.1, 2.0))`	• Applies Gaussian blur with random intensity (σ) to smooth the image.
`T.RandomAutocontrast(p=0.3)`	• Automatically adjusts image contrast with a 30% probability
`T.RandomEqualize(p=0.1)`	• Equalizes the image histogram (balances brightness levels) with a 10% chance.

Random Erasing & Normalize

`ToTensor()`	• Converts the image to a **PyTorch tensor** with values normalized to **[0, 1]**.
`Normalize(...)`	• Standardizes pixel values using the mean and standard deviation of **ImageNet**. • Helps stabilize training and improve convergence.

Bbox Transformation

Since the ground truth bboxes are based on the original image size, we choose to transform the ground truth in train.json and valid.json. On the other hand, the predicted bboxes, which are in 256×256 coordinates, should be transformed back to their original size.

My DigitCocoDataset does not apply flipping, scaling, or shifting transformations because torchvision.transforms cannot adjust the bounding boxes accordingly, which would lead to incorrect ground truth annotations. I turned out to use albumentations for stronger data augmentation, and the dataset class were shown as followed. The transformation would focus on flipping, scaling, and shifting, which were shown important for digit recognition tasks.

AlbumentationsDigitCocoDataset

Methods	Discriptions
`Resize((256, 256))`	• Fix training data size.
`A.HorizontalFlip(p=0.5)`	• Randomly flips the image and bounding boxes horizontally (50% chance). • Increases directional diversity of the digits.
`A.Affine(...)`	• Applies random scaling, translation, rotation, and shear to the image. • Bounding boxes are adjusted accordingly to stay aligned.
`A.RandomFog(p=0.4)`	• Simulates fog to reduce image clarity. • Improves model robustness in low-visibility scenarios.
`A.MotionBlur(blur_limit=3, p=0.1)`	• Adds motion blur to mimic fast movement or camera shake. • Trains the model to handle blurred input.
`A.OneOf([...], p=0.2)`	• Randomly applies grayscale or channel dropout. • Reduces color dependence of the model.
`A.CoarseDropout(...)`	• Randomly masks several rectangular regions in the image. • Helps the model learn to detect partially occluded objects.
`bbox_params=A.BboxParams(...)`	• Ensures transformations are correctly applied to bounding boxes. • Discards boxes with visibility less than 50% after transformation.

Model Architecture & Training Strategy

Model Summary

The digit recognition model is built upon the **Faster R-CNN** framework with a **Feature Pyramid Network (FPN)** to enhance detection performance across multiple scales. The core components of the architecture are as follows:

- **Model Type:** Faster R-CNN with FPN
- **Backbone:**
    1. **Custom ResNet/ResNeXt + FPN Backbone:**

        When the model_type is one of 'resnet18', 'resnet50', 'resnet101', 'resnext50_32x4d', or 'resnext101_32x8d', the model uses a custom backbone created via resnet_fpn_backbone(). This backbone leverages ImageNet-pretrained ResNet or ResNeXt variants for feature extraction.
    2. **Faster R-CNN ResNet-50 FPN V2 (Official Model):**

        This version includes a ResNet-50 + FPN backbone with trainable Batch Normalization layers, which helps adapt better to new datasets during fine-tuning. It uses a convolutional predictor head for classification and bounding box regression, improving accuracy over traditional linear layers.
- **Feature Pyramid Network (FPN)**: FPN is a neural network architecture designed to improve multi-scale object detection, especially for small objects. FPN integrated with the backbone, the FPN enhances the model's ability to detect digits at various scales by constructing a multi-scale feature hierarchy. This is especially important for small object detection, as digits may vary in size across different images.
- **Region Proposal Network (RPN):** The RPN scans the feature maps produced by the FPN and generates region proposals (candidate object locations) by predicting objectness scores and bounding boxes.
- **RoI Heads (Region of Interest Heads)**: These components take the proposed regions and classify them into one of the **11 categories** (category "1" to "10", plus a background class). They also refine the bounding box coordinates for more accurate localization.
- **Customizable Weights:** Allows loading trained checkpoints

Training Setup

Optimizer	`optim.AdamW(filter(lambda p: p.requires_grad, model.parameters()), lr=0.0001, weight_decay=1e-4)`
Scheduler	`optim.lr_scheduler.CosineAnnealingLR( optimizer, T_max = num_epoochs )`
Loss	**Faster R-CNN (from torchvision) internally computes:** • **RPN classification loss (objectness)** • **RPN bbox regression loss**

	• **RoI head classification loss** • **RoI head bbox regression loss** **These are returned in a dictionary `loss_dict`, and I compute the total loss by summing them up.**
LR	**1e-4**
Batch size	Tried on 32 and 16.

Evaluation

I use TorchMetrics' MeanAveragePrecision to compute:

- **mAP@[.5:.95]**: averaged over multiple IoU thresholds from 0.5 to 0.95.
- **mAP@0.5**: IoU threshold fixed at 0.5.

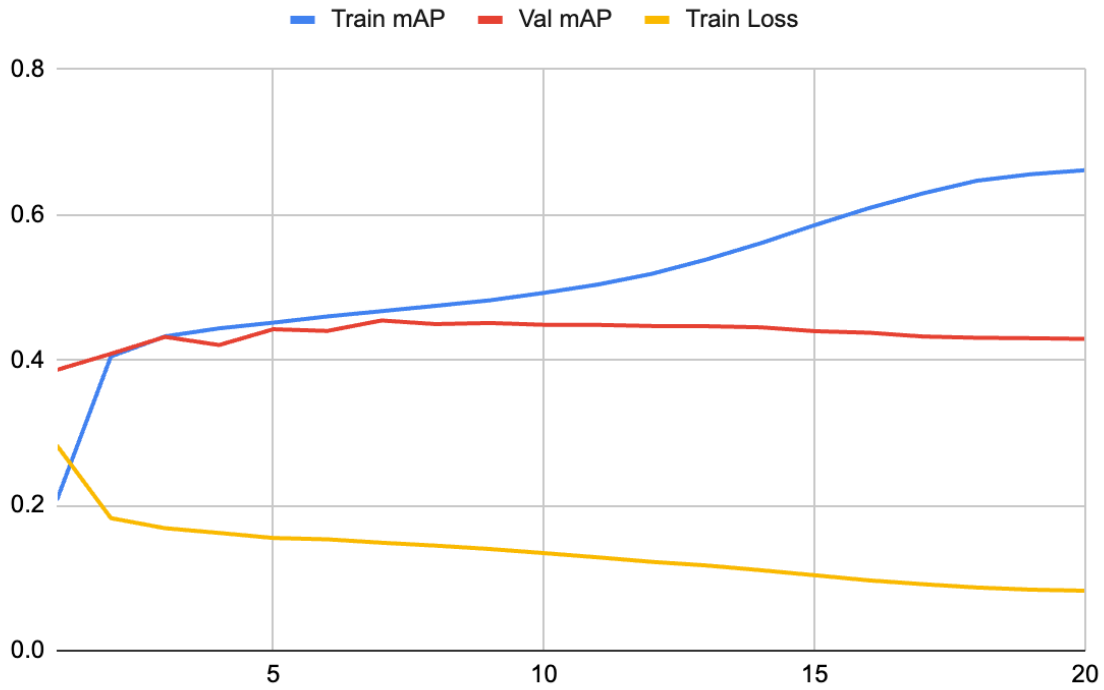
I recorded both values for training and validating phases.

Experiment

I achieved a leaderboard score by training the DigitCocoDataset for 10 epochs with threshold = 0.7. Subsequently, I attempted to train the AlbumentationsDigitCocoDataset, which incorporates more complex data augmentations. However, the training process for this dataset was significantly more time-consuming. As a result, I initialized the training of AlbumentationsDigitCocoDataset using the pre-trained weights obtained from the DigitCocoDataset model.

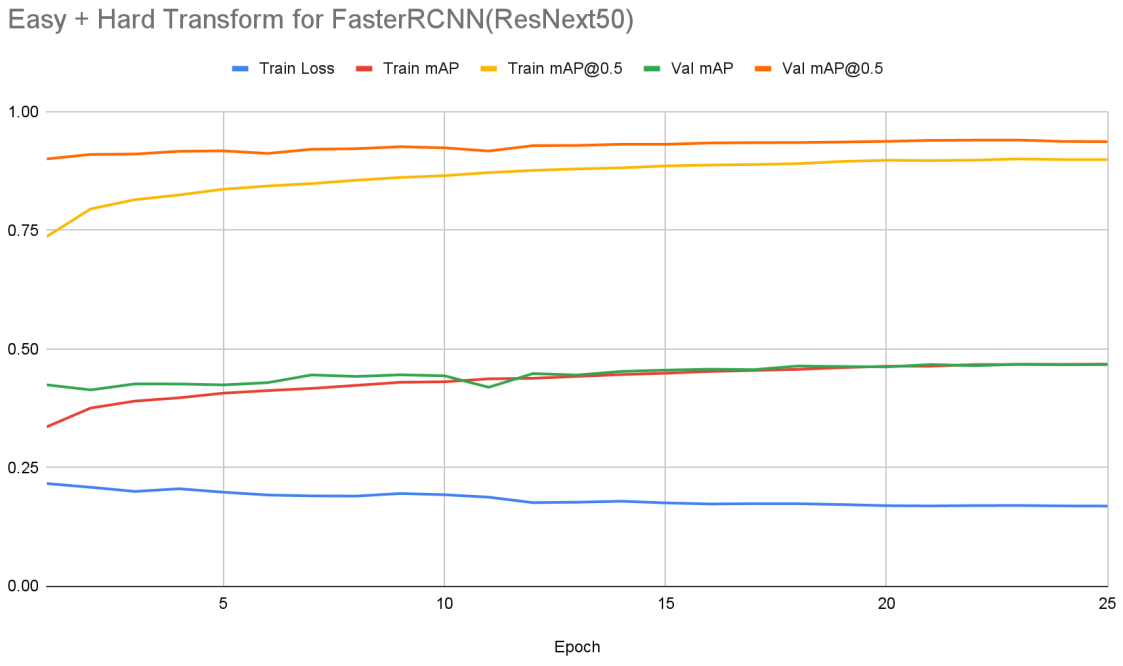
Overfitting Problem

The model was trained on the DigitCocoDataset for 20 epochs with a batch size of 32. As illustrated in the following plot, a noticeable overfitting trend emerges when the number of training epochs exceeds 10, as evidenced by the widening gap between training and validation performance.



Albumentation Transform

I utilized a model pretrained on the DigitCocoDataset for 10 epochs as the backbone and subsequently fine-tuned it using a stronger set of Albumentations-based data augmentation techniques. The results indicated that, while the enhanced transformations effectively mitigated overfitting, the training loss exhibited a significantly slower rate of decline. Ultimately, the model's performance converged to a level comparable to that of the initial pretrained backbon.



References

[1] **Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., & Kalinin, A. A.** (2020). *Albumentations: Fast and Flexible Image Augmentations*. Information, 11(2), 125. https://www.albumentations.ai

[2] PyTorch Core Team. (2024). *Faster R-CNN — Torchvision main documentation*. PyTorch. https://pytorch.org/vision/main/models/faster_rcnn.html

[3] Portions of the code and explanations were assisted by OpenAI's ChatGPT.