Digit detection with faster R-CNN resnet50_fpn_v2

Introduction

This report details the implementation and training of a Faster R-CNN model for digit recognition on the Street View House Numbers (SVHN) dataset. The primary goal is to accurately identify sequences of digits within images. The project addresses two specific tasks:

1. **Task 1: Bounding Box and Class Prediction:** Detecting individual digits within an image, predicting their bounding boxes, and classifying each detected digit (0-9).
2. **Task 2: Full Number Sequence Recognition:** Reconstructing the complete sequence of digits present in the image, based on the detections from Task 1.

The core idea is to leverage a state-of-the-art object detection model, Faster R-CNN, pre-trained on a large dataset (COCO), and fine-tune it specifically for the task of SVHN digit detection. Task 1 directly utilizes the model's output. Task 2 involves a post-processing step where the detected digits from Task 1 are filtered by confidence, sorted spatially (left-to-right), and then concatenated to form the final number sequence. The project utilizes PyTorch and Torchvision, executed within the Google Colab environment.

Github: https://github.com/reson320/VRDL_Homework/tree/main/Homework2

Method

Data Preprocessing

- **Dataset:** The data was split into training, validation, and testing sets, with image files stored in respective directories (train, valid, test). Annotations were provided in COCO JSON format (train.json, valid.json). However, the test.json was missing. Thus, I create it by python script, test.json contains image_id, image_name, width and height.
- **Annotations**: For training and validation, the dataset loader extracts bounding boxes (provided as [x_min, y_min, width, height]) and category IDs for each digit. The category IDs range from 1 to 10, corresponding to digits '0'

through '9'. Bounding boxes are internally converted to the [x_min, y_min, x_max, y_max] format required by the model, ensuring coordinates are clipped within image boundaries. Invalid boxes (zero or negative width/height) are filtered out.

- **Transforms:** Basic data transformation was applied using torchvision.transforms (preferring v2 if available). The primary transform used is ToTensor(), which converts PIL images to PyTorch tensors and scales pixel values to the [0, 1] range. No complex data augmentation techniques (like random cropping, flipping, or color jitter) were explicitly defined in the provided get_transform function for the training set.
- **DataLoaders:** The training loader used shuffling (shuffle=True), while validation and test loaders did not. A batch size of BATCH_SIZE = 8 was used for training.

Model Architecture (Faster R-CNN)

The object detection model used is Faster R-CNN, specifically the **fasterrcnn_resnet50_fpn_v2** implementation from Torchvision, initialized with default pre-trained weights (weights="DEFAULT") learned on the COCO dataset. The architecture consists of the following key components:

- **Backbone (ResNet-50):** A Residual Network with 50 layers serves as the backbone. It processes the input image and extracts a hierarchy of feature maps at different spatial resolutions. These features capture increasingly complex patterns in the image.
- **Neck (FPN - Feature Pyramid Network):** The FPN takes the feature maps generated by different stages of the ResNet-50 backbone and combines them. It builds a pyramid of feature maps with strong semantics at all levels, enabling the model to detect objects (digits, in this case) across a range of scales effectively.
- **Neck (RPN - Region Proposal Network):** The RPN takes the feature maps from the FPN as input. It slides a set of pre-defined anchor boxes over these feature maps. For each anchor, it predicts two things: an 'objectness' score

(probability that the anchor contains *any* object vs. background) and bounding box regression offsets to refine the anchor's position and size. This process generates region proposals – candidate bounding boxes that likely contain objects. The standard RPN from the Torchvision implementation was used without modification.

- **Head (RoI Heads):** The RoI (Region of Interest) Heads take the proposed regions from the RPN and the feature maps from the FPN. Using an operation like RoIAlign, it extracts fixed-size feature vectors for each proposal. These features are then fed into two parallel branches:
    - Classification Head: Predicts the specific class of the object within the proposed region. The original classification head (trained on COCO classes) was replaced by a FastRCNNPredictor. The input dimension (in_features) was automatically inferred from the ResNet-50+FPN backbone, and the output dimension was set to NUM_CLASSES = 11 (10 digit classes + 1 background class).
    - Bounding Box Regression Head: Further refines the bounding box coordinates for each proposal, specific to the predicted class. The standard regression head associated with the FastRCNNPredictor was used

Hyperparameters

- **Batch Size:** 8
- **Learning Rates:** First stage 0.01, second stage 0.001.

  Using ReduceLROnPlateau, the learning rate is reduced by a factor of 0.1 (factor=0.1) if the validation loss does not improve for 2 consecutive epochs (patience=2). The scheduler monitors the average validation loss (mode='min').
- **Number of Epochs:** 10 epochs for both stage1 and 2.
- **Pre-trained Weights:** Training starts from the fasterrcnn_resnet50_fpn_v2

COCO pre-trained weights.

Task 2 Post-processing

Task 2 involves generating the full digit sequence from the raw detections produced by the Faster R-CNN model (Task 1). This is achieved through the following post-processing steps applied to the Task 1 output (pred.json generated from the test set):

1. **Grouping:** Detections are grouped by their image_id.
2. **Filtering:** Detections with a confidence score (score) below a threshold (TASK2_SCORE_THRESHOLD = 0.5) are discarded.
3. **Sorting:** For each image, the remaining high-confidence detections are sorted horizontally based on the x-coordinate of the center of their bounding box (bbox[0] + bbox[2] / 2).
4. **Mapping & Concatenation:** The category_id of each sorted detection is mapped to its corresponding digit character ('0'-'9') using a predefined dictionary. These characters are concatenated in the sorted order to form the predicted number sequence string.
5. **Handling Empty/Missing:** If an image has no detections passing the score threshold after filtering, or if an image ID from the test set annotation file was not present in the model's predictions, its predicted label is set to "-1".
6. **Output:** The final results are saved to a CSV file (pred.csv) containing two columns: image_id and pred_label (the predicted number sequence string or "-1").

- **Results**

The model performance was evaluated based on loss during training , mAP score for Task1 and accuracy for Task2

- **Training Curves:** Displayed a clear decrease in loss and improvement in loss, demonstrating effective learning. In stage 1, which serves as the main training phase, both train and valid loss show significant improvements. In stage 2, the goal was to further refine the model, so the learning rate was reduced to 0.001. After lowering the learning rate, the validation loss slightly improved to

around 0.1890 in the early epochs. However, it seems to be overfitting when it comes to late epochs.

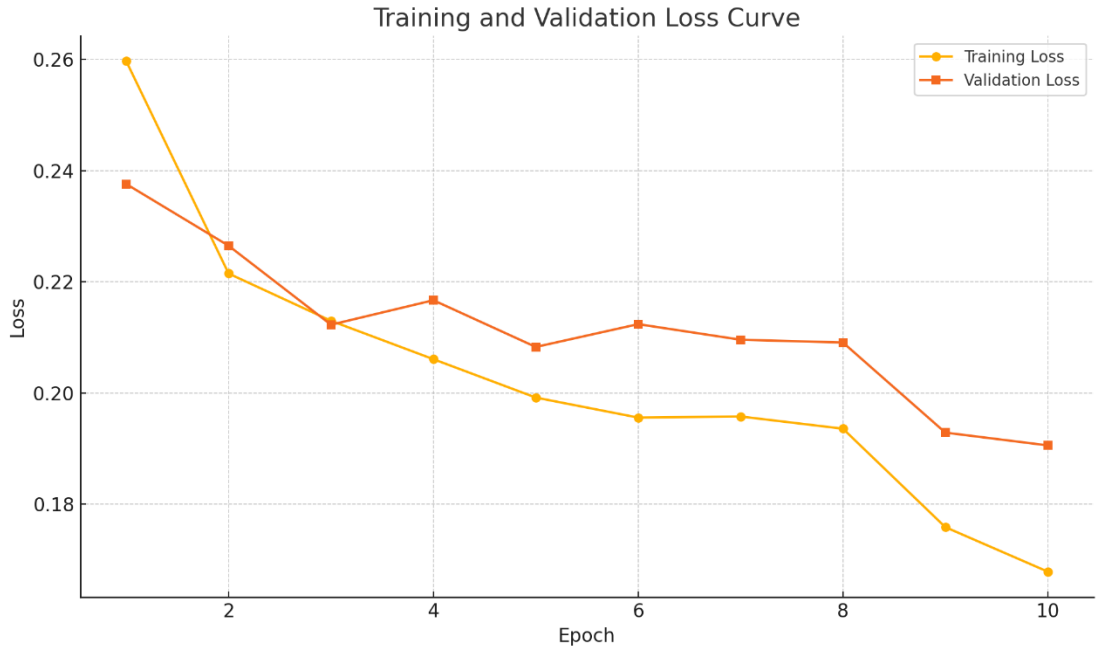

Figure1-training curve at stage 1

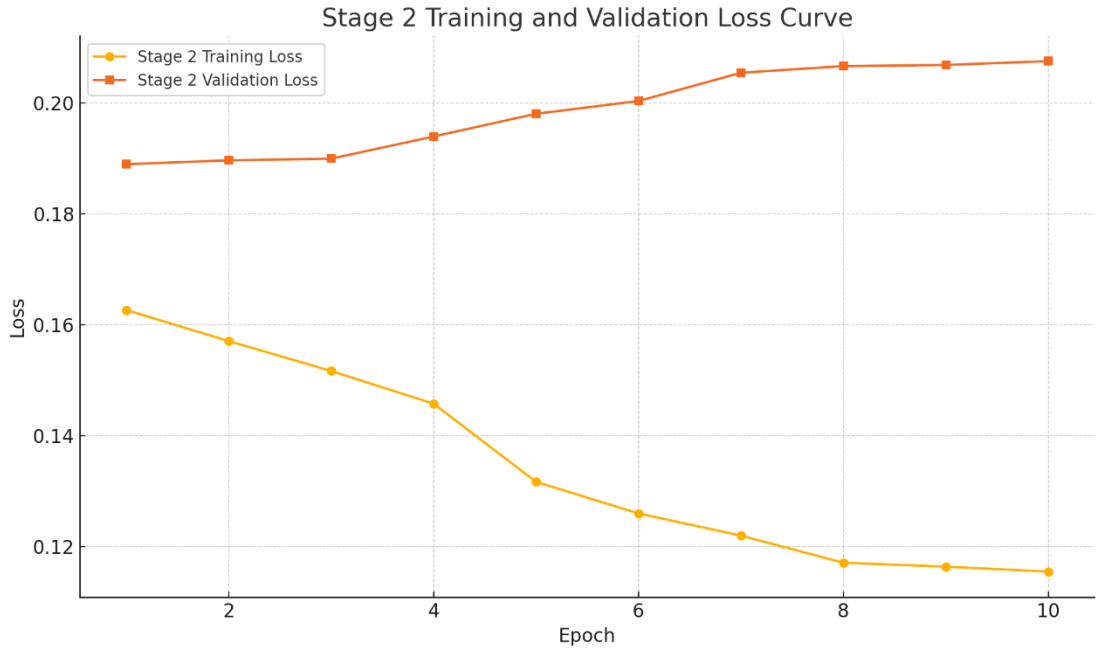

Figure2-training curve at stage 2 (overfitting)

- **Task1 and 2 result:**

Using fasterrcnn_resnet50_fpn as baseline compare the mAP score and accuracy on task1 and task2.(private score)
There might be some problem in fasterrcnn_resnet50_fpn task2 prediction, so I didn't get my task2 score, but I still don't know why.

	Task1	**Task2**
fasterrcnn_resnet50_fpn	**0.370**	**0.0**
fasterrcnn_resnet50_fpn_v2	**0.399**	**0.814**

- **Additional Experiments to Explore Better Performance**

Among various backbone architectures, we chose to explore alternative designs beyond the standard ResNet-50-FPN, with the hypothesis that changing the backbone to a lighter or more advanced architecture might improve performance due to either efficiency or representational capacity.

Hypothesis

- MobileNetV3 would accelerate training and inference with minor trade-offs in accuracy.
- Swin Transformer would improve detection accuracy due to its global attention and hierarchical feature modeling.

Methodology

We replaced the default resnet50 backbone in fasterrcnn_resnet50_fpn_v2 with:

1. mobilenet_v3_large().features as the backbone, followed by FPN.
2. swin_t from timm with added FPN-compatible output layers.

Additionally, the ROI heads and RPN settings were adjusted to match the output channel sizes of the new backbones.

Results and Implications

I had no enough time for training swin_transformer, but I trained mobilenet_v3. Thought the training might faster, the result is not good as fasterrcnn_resnet50_fpn_v2 when they are both converge.

	Task1	Task2
mobilenet_v3	**0.379**	**0.79**
fasterrcnn_resnet50_fpn_v2	**0.399**	**0.814**

References

- *Pytorch-Faster R-CNN*: https://pytorch.org/vision/main/models/faster_rcnn.html
- Github Alan-Lee123/ml100
  [GitHub repository] https://github.com/Alan-Lee123/ml100
- Github: Davelbit-*DL-CV*
  [GitHub repository] https://github.com/davelbit/DL-CV