COL/JRL780 - Computer Vision
Assignment 3: Deep Learning for Object Detection

Deadline: 08 May 2025

Introduction

In this assignment, you will explore state-of-the-art object detection models and fine-tuning strategies to enhance detection performance. You will work with pre-trained models, experiment with zero-shot detection, prompt tuning strategies, and develop a competitive detection solution using any architecture of your choice. This hands-on experience will deepen your understanding of advanced object detection techniques and their practical applications.

Dataset

The dataset provided is a subset of the Foggy Cityscapes dataset with COCO-style annotations. The dataset is split into train and validation sets, and you are required to report all your evaluation scores on the validation set. Find the dataset here: link

Task 1: Evaluating and Fine-Tuning Pretrained Deformable DETR (40 points)

Objective: Evaluate the performance of a pre-trained Deformable DETR model on the provided dataset and improve detection performance via fine-tuning.

Instructions

1. **Evaluation of Pretrained Model:**
    - Load a pre-trained Deformable DETR model (checkpoint).
    - Run inference on the dataset and qualitatively analyze the detection results using visualizations.
    - Perform a quantitative evaluation by generating a precision-recall table at different thresholds and computing the mean Average Precision (mAP) for the pretrained model.

2. **Fine-Tuning:**
    - Implement a training loop to fine-tune the model on the dataset.
    - Use an appropriate optimizer (e.g., AdamW) and the **SetCriterion** loss from DETR for training. (For details on SetCriterion, refer to the original DETR repository: DeTr.)
    - **Experiment 1:** Fine-tune the full model (update all layers during training).
    - **Experiment 2:** Fine-tune only the decoder.
    - **Experiment 3:** Fine-tune only the encoder.
    - Save model checkpoints and compare detection performance across all experiments.

Deliverables:

- **Code:** Submit your complete code for both evaluation and all fine-tuning experiments, along with a bash script to run your code.
  Example usage: `task1.sh <subtask> <experiment> <mode> <input_dir>`
- **Report:**
  - An explanation of your model setup and training process.
  - Training Curves: Display training curves (e.g., loss vs. epoch) for each experiment.
  - Quantitative Evaluation: Compute the mean Average Precision (mAP) for the pretrained model as well as for each fine-tuning strategy. mAP is the precision averaged across confidence thresholds, classes, and iou thresholds. You may use the pycocotools library for this purpose. Refer this link to better understand mAP.
  - Qualitative analysis with visualizations of detection results.
  - A discussion comparing the effects of fine-tuning the full model versus fine-tuning only the decoder versus fine-tuning only the encoder layers, including insights and challenges encountered.

Task 2: Zero-Shot Evaluation and Prompt Tuning with Grounding DINO (40 points)

Objective: Evaluate the zero-shot object detection capabilities of a pre-trained Grounding DINO model. Implement prompt tuning and analyze how different prompt tuning strategies influence detection performance.

Official G-DINO repo: Grounding DINO **Other useful links:** Open-GroundingDINO, MMDetection, CoOp, CoCoOp

Instructions

1. **Model Setup and Inference:**
   - Load a pre-trained Grounding DINO model.
   - Run inference on the provided dataset using a baseline text prompt (e.g., "an apple") and analyze the detection outputs using visualizations.
2. **Prompt Tuning:**
   - Modify the text prompt inputs by introducing learnable prompt embeddings.
   - Experiment with different prompt tuning strategies to determine which approach works best for the given dataset.
   - Run inference for your chosen prompt tuning strategy on the same set of images and record the detection outputs, noting differences in precision and the quality of bounding boxes and classifications.

Deliverables:

- **Code:** Submit your complete code for loading the model, performing zero-shot inference, and generating both quantitative and qualitative evaluation outputs. Include a bash script to run your code.
  Example usage: `task2.sh <subtask> <mode> <input_dir>`
- **Report:**
  - An explanation of your experimental setup and the rationale behind your chosen prompt tuning strategy.
  - Quantitative Evaluation: Perform the same analysis as Task 1 for the baseline prompt and for your prompt tuning strategy.
  - Qualitative analysis with visualizations of detection results corresponding to your prompt tuning strategy.
  - A discussion of your findings, including insights on how prompt tuning impacts detection performance and any challenges encountered during the process.

Task 3: Competitive Challenge — Achieving the Best Performance on a Hidden Test Set (20 points)

Objective: Develop an object detection solution using any model/architecture of your choice to achieve the best performance on a hidden test set that is similar to the provided dataset.

Instructions

1. **Model Selection and Implementation:**
    - You are free to choose any object detection model or architecture (e.g., YOLOv8, Faster R-CNN, DETR variants, etc.) that you believe can deliver the best performance.
    - You do not need to implement the model from scratch; you can use available pretrained checkpoints and fine-tune them on the provided dataset.
    - Experiment with data augmentation, hyperparameter tuning, and transfer learning techniques to optimize performance.
2. **Evaluation:**
    - Since you do not have access to the hidden test set, evaluate your model on the provided validation set.
    - Generate both quantitative and qualitative evaluations.
    - Compare your results with the baseline performance from Tasks 1 and 2, and discuss the improvements and trade-offs.

Deliverables:

- **Code:** Submit your complete code (including any training, evaluation, and preprocessing steps) along with a bash script to run your code.
  Example usage: `task3.sh <mode> <input_dir>`
- **Report:**
    - An explanation of your chosen model/architecture and the rationale behind your selection.
    - Details on your fine-tuning process, hyperparameter tuning, and any modifications or improvements implemented.
    - Quantitative Evaluation: Perform the same analysis as Tasks 1 and 2.
    - Qualitative analysis with visualizations of detection results.
    - A discussion of insights gained, challenges encountered, and comparisons with baseline results from Tasks 1 and 2.

Submission Guidelines

Submit a PDF report containing all deliverables.

Zip the following files (e.g., `entrynumber.zip`) and upload on Moodle:

- All source code files (`.py` files)
- Bash scripts: `task1.sh`, `task2.sh`, `task3.sh`

- Requirements files: `requirements1.txt`, `requirements2.txt`, `requirements3.txt`

Upload all your checkpoints to a cloud storage service (e.g., Google Drive, OneDrive) and include the link in your report with appropriate naming (task, subtask, experiment).

Evaluation

We will install the required packages from the requirements files. Then, we will run the task scripts using the following command line parameters:

- `<task>`: 1/2/3
- `<subtask>`: 1/2 (for Tasks 1 and 2)
- `<experiment>`: 1/2/3 (for Task 1)
- `<mode>`: train/infer
- `<input_dir>`: Path to the directory containing input data in COCO format

Ensure that your code creates a directory `trained_checkpoints` and saves all checkpoints there with appropriate naming. When inferring, your code must create a directory `outputs/<task_num>` and save JSON files in COCO format with appropriate naming.

Some Tips

- Start early to have ample time for experimentation. Note that Kaggle/Colab provide a fixed number of GPU hours per week.
- Use extensive visualizations to ensure everything is working correctly and avoid realizing mistakes at the end.
- Experiment with hyperparameter tuning, as it is crucial for achieving good performance.