Inmind Academy

Computer Vision Assignment

Ref: -

Date: 8-Aug-2024

Revision: 2

Powered by **inmind.ai**

Table of Contents

GUIDELINES

This project will focus on object detection tasks for computer vision. The deadline for this project is Friday 23th of August 2024. The presentation is primarily during the academy session from 2 PM to 5 PM, students are required to create a private GitHub repository and share it with the instructor (View access) by the **16th of August 2024**. Students should develop their solution on GitHub and continuously update the repository. Commits done after Friday 23th of August 2024 at 1:00 PM will not be evaluated. For students who cannot attend the presentation please **contact the instructor before the 16th of August 2024** to schedule a meeting accordingly (Only acceptable for urgent cases).

The cleanliness of the students' GitHub commits will be taken into consideration, therefore try to keep them as neat as possible (Follow a single **'best practice'** way of writing commits and stick to it, additionally, commits shouldn't be very large)

Plagiarism is not tolerated.

ASSIGNMENT DESCRIPTION

The dataset provided a collection of labeled images with labels: **Forklift, Rack, Crate, Floor, Railing, Pallet, Stillage, iwhub and dolly** using BMW JSON format. You are requested to:

1. Load the dataset and labels (Using PyTorch DataLoader)
2. Write a function to visualize some of the labeled images (With bboxes)
3. Check the possibility of augmenting the dataset (Albumentations)
4. Split the Training dataset into **train and validation**.
5. Train an object detection model based on **YOLOv7** (You can use YOLOv5 if you are facing hardware issues)
6. Evaluate the model on the **Testing dataset** and relaunch the training with **different hyperparameters.**
7. Train a custom semantic segmentation model using **Pytorch** (The model **code should be written by the student** and available in the GitHub repository; the

architecture can be inspired by an existing model, but this needs to be **clearly stated,** *bonus points if the model is directly inspired by a research paper*)

8. Evaluate the semantic segmentation model on the **Testing dataset** and relaunch the training with **different hyperparameters**.
9. Use Tensorboard to visualize the evolution of both models' metrics during training.
10. **Interpret** the difference in results between the object detection models and the semantic segmentation models separately (in terms of **metrics**: accuracy, iou, **training time**, **inference speed**, **hyperparams**, etc .)
11. **Export both models** into any **inference model type** (ONNX, TensorRT, OpenVino) and test that the inference works and that it is faster than performing it in PyTorch directly.
12. Visualize both networks using Netron
13. **Create an Inference API** that runs on CPU using the exported models (Note: in case you exported the model into ONNX or OpenVino you can use the exported model on CPU, TensorRT will not run on CPU thus you need to use the raw model) with the following endpoints:
    a. Model listing endpoint (return available models)
    b. Segmentation Inference endpoint: takes an image as input and returns the semantic segmentation version of the image
    c. Bbox Inference endpoint: takes an image as input and returns a **JSON** response containing available bounding boxes with respective class and accuracy.
    d. Bbox Inference endpoint: takes an image and returns the **same image with bounding boxes overlayed on it** (Meaning that the endpoint should return an image with bounding boxes drawn on this image).
    e. Dockerize the API into a runnable container

Optional Tasks:

- **Ensemble Models:** Use ensemble methods to combine the results of multiple object detection or segmentation models to improve accuracy.

- **Transfer Learning**: Apply transfer learning from a pre-trained model on a related dataset and fine-tune it on the given dataset.
- **Real-time Inference:** Implement a **real-time inference pipeline** that processes live video feeds and displays the results.

WHAT'S IMPORTANT

Develop a high-performance model optimized for the available hardware, ensuring that the implementation is both efficient and adaptable. Your code should be clean, modular, and maintainable, with an emphasis on reusability and flexibility. Parameter configurations should be easily adjustable to facilitate experimentation and fine-tuning.

Please avoid submitting a Jupyter notebook with 10000 lines of code or a monolithic Python file. Organize your work thoughtfully, breaking it down into clear, manageable components.

And Have Fun.