Sarang Deb Saha

Submission Report

Task 1: The primary objective of this task was to download the DeepFashion dataset, create a manageable sample size of 500 images, and prepare this subset for a machine learning project using YOLO (You Only Look Once) for object detection. The specific requirements included filtering classes with sufficient representation and organising the data into specific training, validation, and test splits.

1. Downloading the .tar file from the mentioned s3 bucket took me a lot of time due to the non-availability of reliable internet.

2. Data Processing: Loading Annotations: Annotations were loaded from the JSON files (instances_train2024.json and instances_val2024.json) which contain information about image filenames, file paths, bounding boxes, and category IDs.

   Filtering by Class Frequency: Classes with fewer than 50 images across the dataset were filtered out using the filter_by_class_frequency function. This ensures that only classes with sufficient data for training are included.

   Splitting Data: The dataset was split into train, validation, and test sets with a ratio of 7:2:1 using the split_data function. This facilitates proper evaluation of the model's performance.

3. YOLO Format Conversion: Writing YOLO Annotations: Annotations were converted to YOLO format, where each annotation file corresponds to an image and contains lines representing bounding box annotations in YOLO format. The write_yolo_annotations function accomplishes this task.
4. The code is structured in a modular manner, with functions for each specific task, enhancing readability and maintainability.
   The main function orchestrates the entire process, from downloading the dataset to converting annotations to YOLO format.
5. The final YOLO-formatted sample dataset is stored in the specified output directory (output_base). After completion, temporary files extracted during processing are removed using shutil.rmtree.

Task 2: Modify the model and training code from the given github repo accordingly:

a. Add another head to YOLOv9-c model architecture so that you can predict both detection (bounding box) and instance segmentation mask at same time, along with category and confidence scores.

   The objective of the modification was to adapt the YOLOv9 architecture for instance segmentation tasks, specifically targeting the DeepFashion dataset, which contains 50 classes/categories.

   Changes Made-

   1. Parameters:
   nc: The number of classes was updated to 50 to match the DeepFashion dataset.

2. Backbone:
The backbone architecture remains unchanged from the original YOLOv9.

3. Detection Head:
The detection head was retained from the original YOLOv9.
DualDDetect is utilised for object detection, considering feature maps from different levels of the backbone.

4. Instance Segmentation Head:
- Additional modifications were made to the instance segmentation head to accommodate the requirements of instance segmentation tasks, including mask prediction, category scores, and confidence scores.
- Upsampling and concatenation layers are employed to combine features from different levels of the backbone.
- Multiple RepNCSPELAN4 blocks are used for feature extraction and refinement.
- The final convolutional layer predicts masks with a sigmoid activation function (nn.Sigmoid).
- An additional convolutional layer is introduced to predict category scores for each instance.
- Another convolutional layer predicts confidence scores, representing the probability of each predicted instance being correct.

b. Modify the training script to take a single argument which contains a yaml config file that has all the arguments and hyperparameters.

Changes Made-

1. Argument Parsing Function: The objective of the modifications was to enhance the training script to accept a single argument containing a YAML config file. This config file encapsulates all the necessary hyperparameters and arguments required for training.

Added an argument parsing function using the argparse module to accept a single argument --config, which specifies the path to the config YAML file.
The --config argument is required for the script to execute successfully.

2. Configuration Loading: Implemented a load_config function to read the YAML configuration file and return its contents as a dictionary.
The load_config function ensures that the hyperparameters and options specified in the YAML file are correctly loaded and accessible within the script.