Approach for solving the problem

Step 1: Dataset Preparation

The first step involves preparing the dataset by creating a smaller sample for faster prototyping and experimentation. This is achieved by selecting a subset of images from the DeepFashion dataset and dividing them into train, validation, and test splits. The approach taken here is as follows:

Reading the Dataset: The script assumes that the DeepFashion dataset is already downloaded and extracted. It navigates through the directory structure to locate the images folder containing the train, validation, and test splits.

Sampling Images: It randomly shuffles the list of image filenames in the train split and selects a subset of images to create a sample dataset. This ensures randomness in the selection process.

Splitting the Dataset: The sample dataset is divided into train, validation, and test splits based on predefined ratios (7:2:1 in this case).

Copying Images: Images are copied from the original dataset splits to corresponding splits in the sample dataset.

Step 2: Dataset Conversion

In this step, the sample dataset is converted into the required format for training the YOLOv9 model. The conversion involves two main steps:

MSCOCO Format: If necessary, the dataset is converted into MSCOCO format, which is a widely used format for object detection and instance segmentation tasks. This format organizes annotations in a JSON file containing information about images, categories, and annotations.

YOLO Format: The MSCOCO-formatted dataset is then converted into the YOLO format, which is specific to the YOLO (You Only Look Once) object detection framework. YOLO format typically consists of text files for each image containing object bounding box coordinates and class labels.

Step 3: Model Training

The YOLOv9 model is trained on the sample dataset using the modified training script. Instead of passing hyperparameters through command-line arguments, a config file is used to store all the parameters. This approach offers flexibility and ease of modification for experimenting with different hyperparameters.

Step 4: Performance Metrics

Once the model is trained, performance metrics such as mean Average Precision (mAP) are calculated on the validation and test splits. mAP is a common metric for evaluating object detection models as it considers both precision and recall across different IoU thresholds.

Step 5: Model Optimization

The trained model is optimized for deployment using ONNX and OpenVINO formats. ONNX allows interoperability between different deep learning frameworks, while OpenVINO enables efficient deployment on Intel hardware. Visualizations of model predictions are generated for qualitative assessment.

Step 6: Latency and Throughput Comparison

The latency (inference time) and throughput (number of predictions per unit time) of the PyTorch, ONNX, and OpenVINO models are compared on the test set. This comparison helps in understanding the performance trade-offs between different deployment formats.

Step 7: Dockerization

Finally, a Dockerfile is created to encapsulate the model training and inference process. The Dockerfile includes support for CUDA (for GPU acceleration) and OpenVINO (for CPU inference on Intel hardware), enabling seamless deployment across different environments.