CNN model to extract product attributes from image

Objective: Show a basic architecture and implementation of a **CV (computer vision) CNN** model to extract product attributes by using object detection, by training custom product image data on large pre-trained CNN models which can then be improved on by further tuning

1. This initial model shows how to use **CNN** models trained on public datasets to perform object detection, tagging, categorizing to identify product attributes for improving customer engagement and product recommendations for sales
2. The tags or labels are effectively the product attributes
3. Model can be trained on custom data, on top of the pre-trained model
4. Reinforcement Learning can then be used by freezing the good model layers and re-trained on new image data in an iterative improvement loop

Model Used: Used a pre-trained **Faster R-CNN model with a ResNet-50 backbone** and trained it on **COCO dataset**. There are many other models that can be used (YOLO, SSD, etc)

Reason: Both are popular and widely used models and datasets, with COCO having a good collection of image object labels which can be used for tagging new images.

Basic model terms:

1. **R-CNN:** Regions with Convolutional Neural Networks. It uses a region proposal algorithm to generate 2000 region proposals per image, which are then classified by a CNN
2. **Faster R-CNN:** improves on R-CNN by sharing convolutional features across region proposals, reducing computation time
3. **ResNet-50:** 50-layer deep residual network which has 50 layers that perform convolution, batch normalization, and activation operations
4. Each layer itself has Input Layer, Convolutional Layers, Residual Blocks, Pooling Layers, Fully Connected Layers, Output Layer. This is a typical CNN model architecture
5. **COCO (Common Objects in Context):** large-scale object detection, segmentation, and captioning dataset. It contains over 200,000 labeled images with 80 common object categories such as person, car, chair, etc. It has separate, training, validation and test sets

Overall model training/ Code overview:

1. Using **2 RTX4090 GPU's with 48 GB RAM** each. Training takes ~ 20 mins
2. Trained for simple **5 epochs on 2000 samples** (out of ~ 118,000)
    - All model parameters are in the code
3. Used small samples from the training set for both training and test, with distinct independent ranges to avoid data leakage / mixing data
    - This was due to test set data files giving lot of errors like size differences, etc, need to fix it before using
4. **Training loss reduces from 0.72 to 0.46**. So model is good and can be improved
5. **Test Loss is 0.92, high**, can be improved by using larger samples and more epochs

6. **Evaluating a random sample** image from test data shows decent accuracy in identifying the object labels from the bounding boxes. Can be improved by training
7. The object labels are effectively the product attributes which can be **written to any database** for downstream operations to build customer sales or engagement related models, once customer interaction lifecycle data is attached to image product attributes

Improvements:

1. The model is basic and can be improved in various ways – different pre-trained models, custom dataset, more data, hyperparameter tuning, etc
2. If we train and test on entire dataset, results will definitely improve. Needs overnight run
3. Need to implement DataParallel functionality to ensure pytorch is using all GPU's for training (ResNet-50 model allows this) to see if it makes any difference in speed

Sample Image with bounding box and labels(attributes):