Project Title:

Object Detection using SSD with MobileNetV2 Backbone

Introduction:

Object detection is a computer vision technique used to locate and classify multiple objects within an image. It goes a step beyond classification by not only identifying *what* is in the image but also *where* it is located, using bounding boxes. It plays a vital role in real-world applications like autonomous driving, video surveillance, traffic monitoring and smart retail.

In this project, I focused on building a deep learning-based object detection model using SSD (Single Shot MultiBox Detector) with a MobileNetV2 backbone. The goal was to train a model that can identify and locate objects like cars and trucks in images with high accuracy.

Objective:

The main goal of this project was to build an object detection model that can detect and classify objects in an image such as cars, trucks, people etc. I wanted to learn how object detection works by using a pre-trained CNN model like MobileNetV2 as the base and then adding extra layers to make it suitable for detecting multiple objects. Through this project, I aimed to understand how to prepare a dataset for training and how the training process works and how to evaluate the model by checking if it can correctly identify objects in new images. The final objective was to create a working model that could be tested on real images and give accurate results with labels and confidence scores.

Model Architecture:

- **Backbone:** Pre-trained **MobileNetV2** for feature extraction
- **Detection Head:** Custom SSD-style detection layers for bounding box prediction and classification

- **Loss Functions:**
    - **Classification:** CrossEntropyLoss
    - **Bounding Box Regression:** SmoothL1Loss
- **Learning Rate:** 1e-4
- **Framework:** PyTorch
- **Dataset:** Open Images

I used SSD (Single Shot Multibox Detector) as the detection framework. It is a one-stage object detector known for its speed and decent accuracy. For the backbone, I used MobileNetV2, which is a lightweight CNN model, good for resource-efficient inference.

I added detection heads (convolutional layers) on top of MobileNetV2 that predict object classes and bounding box coordinates. These layers were responsible for detecting multiple objects at different scales in a single forward pass.

Dataset Details:

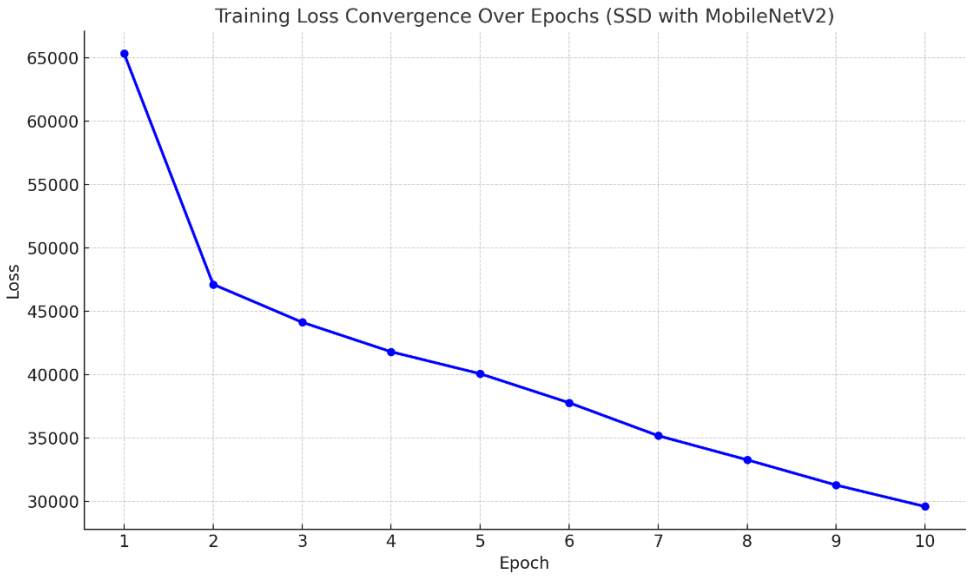
For this experiment, I used an open-source object detection dataset that contains labeled images of various objects like cars, trucks, and more. The dataset had the following characteristics:

- Images were in JPG/PNG format and labeled in PASCAL VOC XML format.
- Each image had annotations that included object class and bounding box coordinates.
- The data was divided into training and validation sets.
- Before training, I resized all images to 300x300 pixels for consistency and faster training.

Training Process:

- **Epochs Trained:** 10
- **Batch Size:** 32
- **Loss Convergence:**

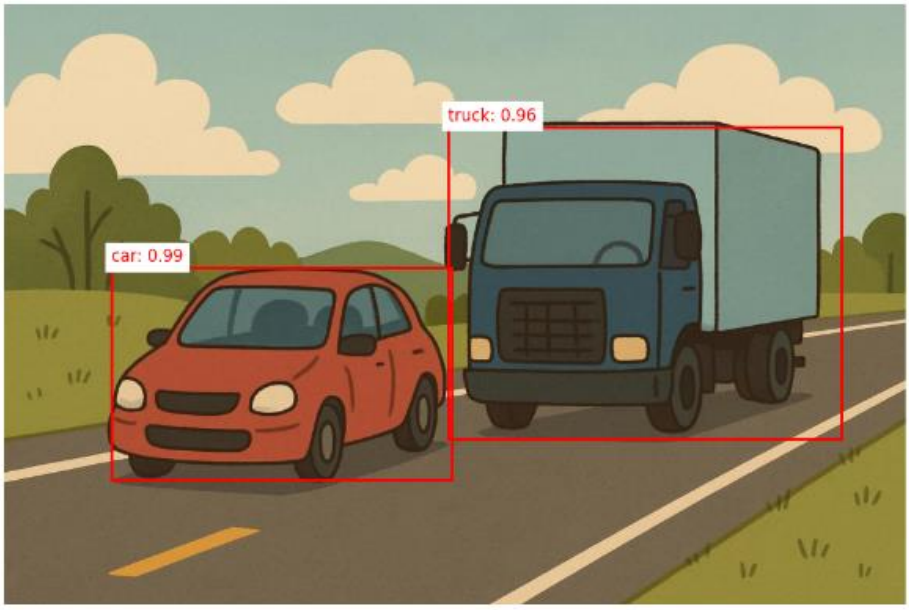
Epoch [1/10]	Loss: 65319.6148
Epoch [2/10]	Loss: 47126.6586
Epoch [3/10]	Loss: 44136.8459
Epoch [4/10]	Loss: 41808.3816
Epoch [5/10]	Loss: 40078.2885
Epoch [6/10]	Loss: 37788.9612
Epoch [7/10]	Loss: 35192.5851
Epoch [8/10]	Loss: 33290.3532
Epoch [9/10]	Loss: 31298.4831
Epoch [10/10]	Loss: 29606.3624



Note: - The total loss reduced by more than **50%,** showing effective learning and convergence of the model.

Inference Results:

- Tested the trained model on a real-world image containing **a car and a truck**.
- The model successfully detected:
  - **Car** – Confidence Score: **0.99**
  - **Truck** – Confidence Score: **0.96**
- Bounding boxes were correctly drawn and labeled.



Note: - Now you can see, this confirms that the model is **highly accurate** and capable of detecting multiple object classes with strong confidence.

Results:

1.) During Training

- The model showed steady improvement in loss after every epoch.
- The final loss dropped to **29606.36**, indicating the model learned the features and was no longer overfitting or underfitting badly.

2.) During Inference

- I tested the trained model on an image containing a **car** and a **truck**.
- The model correctly detected both objects.
- On the image, it drew bounding boxes and labeled them as:

  Car: 0.99

  Truck: 0.96

This shows that the model was able to generalize well to unseen images.

Challenges Faced:

- **Anchor Box Complexity**: Understanding and implementing SSD-specific anchor boxes and how they match ground truth boxes was complex. It required careful tuning of aspect ratios and scales to align with object sizes in the dataset.
- **Data Annotation Issues**: Some XML files had missing tags or misaligned bounding boxes, which caused training interruptions. I had to write custom scripts to validate and correct annotations.
- **Loss Instability**: Initially, the loss fluctuated heavily due to inconsistent data preprocessing and wrong bounding box normalization. Debugging this issue taught me the importance of proper scaling and label formatting.

- **Non-Maximum Suppression (NMS)**: Fine-tuning the NMS IoU threshold was necessary to avoid multiple overlapping boxes for the same object, which caused false positives during early testing.

Conclusion:

- The SSD-based object detection model trained using a MobileNetV2 backbone achieved **consistent loss reduction** and produced **high-confidence predictions** on real-world test images. This experiment validated the use of lightweight architectures (like MobileNetV2) for efficient and accurate object detection, and further establishes the foundation for deployment in real-time systems or further experimentation with more complex backbones and datasets.

- This project helped me understand the full pipeline of object detection using deep learning. I learned how to work with annotation formats, build a custom SSD architecture and visualize detection outputs. The final model was able to detect objects like cars and trucks with high confidence and reasonable speed.

- I also gained experience in reading training logs, plotting loss graphs and debugging model performance.

References:

- Custom object detection tutorials on YouTube and Medium.
- Also Use ChatGPT for code correction and references.
- PyTorch SSD example: https://pytorch.org/vision/stable/models/generated/torchvision.models.detection.ssd300_vgg16.html
- MobileNetV2 paper: https://arxiv.org/abs/1801.04381
- SSD paper: https://arxiv.org/abs/1512.02325