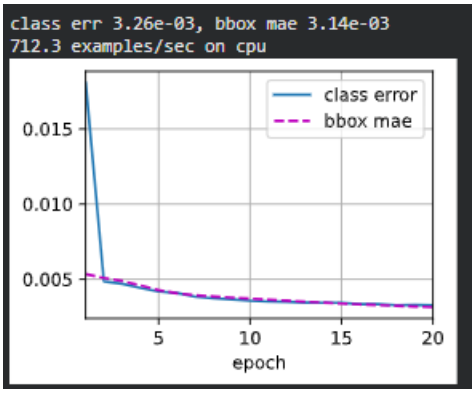
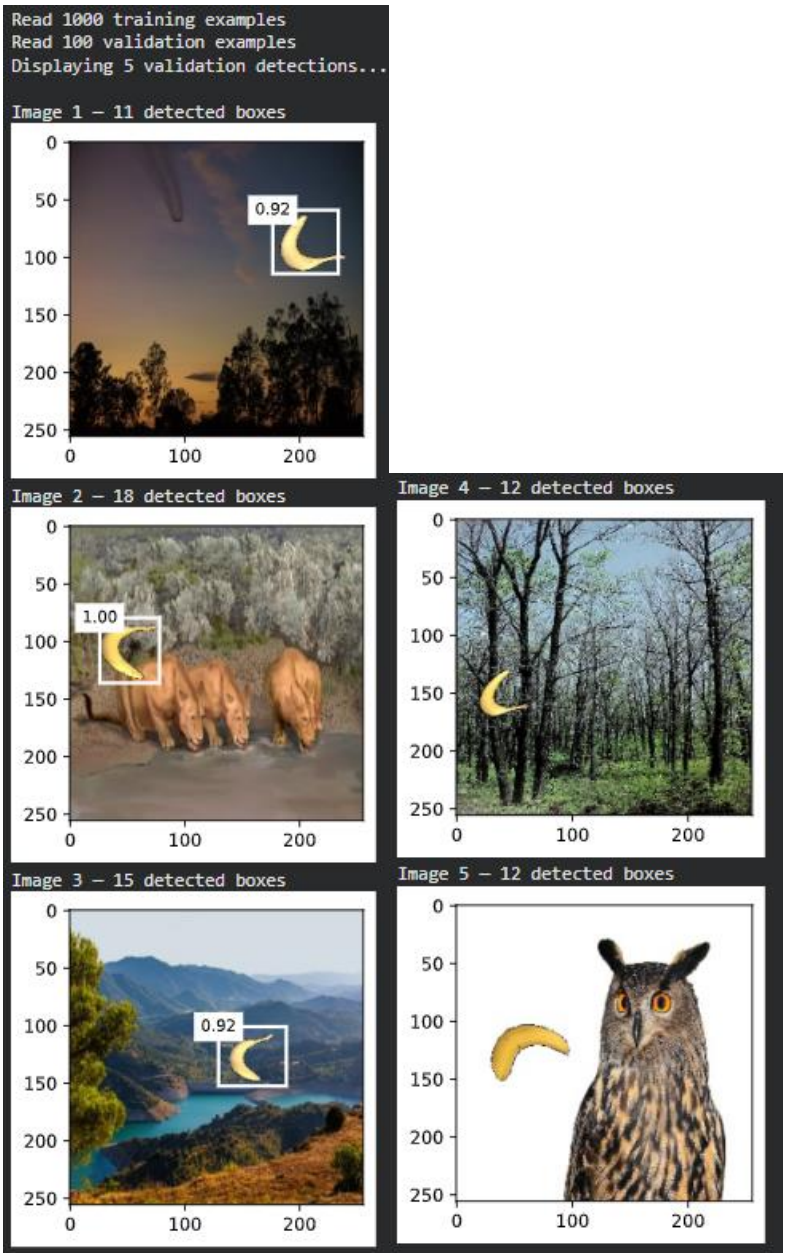
Project 3 Report

Part 1

Banana Detection Samples



Training Loss Curves (Class Error and Bounding Box MAE)



For this part, I trained a small SSD-style object detector (TinySSD) on the D2L banana dataset. The dataset has 1,000 training images and 100 validation images, each containing one banana pasted onto different backgrounds, sizes, and locations. I first loaded the dataset with a custom BananasDataset class and visualized a batch to make sure the ground-truth boxes lined up with the bananas.

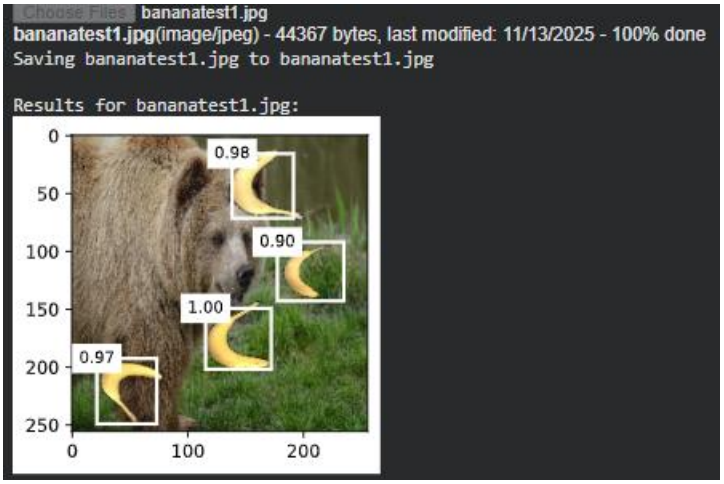
The model architecture follows the standard SSD idea. A small CNN extracts features at multiple scales, and from those feature maps the model creates anchor boxes of different sizes. For each anchor, the network predicts whether it's a banana or background and also predicts four

numbers used to adjust the bounding box. The loss combines cross-entropy for the class prediction and L1 loss for the box offsets.

I trained the model for 20 epochs on CPU using SGD. The training plot shows the class error and bounding box MAE dropping quickly and leveling out at very low values (around 3e-3). This means the model learned the dataset well.

The sample detections show that the model reliably finds the banana in a variety of new backgrounds and usually assigns high confidence scores (0.9–1.0). There are some limitations, though. For example, in the owl image the model perfectly detects the banana but ignores the owl entirely. This happens because the dataset only teaches the model one class: banana. Everything else is treated as "background."

Overall, Part 1 shows that TinySSD can learn this simple one class detection task very well, but it also highlights that the model doesn't learn anything about other objects or more complex scenes.

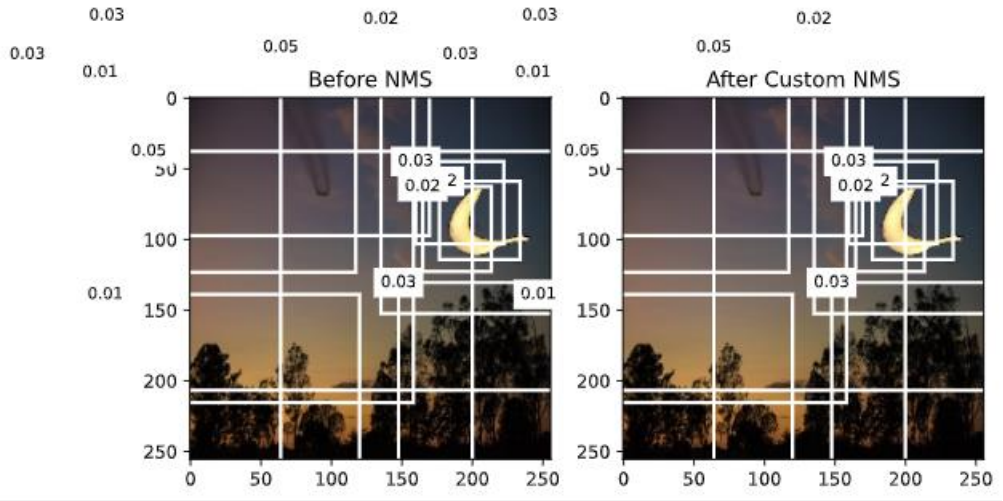


These are my test images.

Part 2

```
Read 1000 training examples
Read 100 validation examples
Custom NMS kept: 11
TorchVision NMS kept: 11
Indices identical? True
```

Custom NMS vs TorchVision NMS Results



Visualization of Detections Before and After Non-Maximum Suppression

In this part I wrote my own Non-Maximum Suppression (NMS) function and compared it to PyTorch's built-in torchvision.ops.nms. NMS is a cleanup step used after the detector makes predictions. The idea is simple: keep the highest-scoring box and remove any other boxes that overlap too much with it (based on IoU). Then move on to the next box and repeat.

My custom version sorts boxes by score, checks IoU between them, and removes any with too much overlap. I also wrote a small IoU function for comparing two boxes. To verify that my code worked, I ran both my NMS and PyTorch's NMS on the same SSD outputs. Both methods kept exactly 11 boxes, and the kept-index lists were identical. That confirms my implementation matches the official one.

I also visualized the predictions before and after NMS. Before NMS, there are a lot of overlapping boxes around the banana. After NMS, most of that clutter disappears and only the important boxes remain. In this example the difference is subtle because the boxes don't overlap that much and some scores are low, so only a few boxes get removed. This shows that NMS depends heavily on the IoU threshold and how crowded the predictions are.

Overall, Part 2 shows that my NMS works the same as PyTorch's version and that NMS is necessary for turning messy SSD output into clean, readable detections. But it also shows that NMS is a simple rule-based method, and in crowded scenes it can accidentally remove valid detections.

Part 3

Baseline HOI Prediction with BLIP-2

Refined Prompt HOI Prediction with BLIP-2

In this part I switched from normal object detection to human–object interaction (HOI) analysis using a vision-language model. Instead of training a new network, I used the open-source BLIP-2 model in a zero-shot setting. The idea is simple: give the model an image and a prompt, and it will describe what the person is doing with the objects, such as or .

For testing, I used an image of a police officer on a horse. With the basic prompt "Describe the human-object interactions in this image," the model responded with a full sentence: "a man is standing on a horse in front of a tree." It understood the main interaction (the person and the horse), but the description wasn't in the HOI label style we needed.

To improve this, I used a more specific prompt: "List all interactions in format, such as or . Only list interactions." With this refined prompt, the model gave a more useful output like " ." It correctly identified the main interaction in the format we wanted, though it also added an extra unnecessary token and still missed other possible actions in the scene.

These results show that BLIP-2 can recognize interactions without training, but its accuracy depends heavily on the prompt. A general prompt gives a loose description, while a structured prompt pushes the model toward HOI-style answers. Even with better prompts, the output can still be incomplete or slightly off. To get more reliable HOI predictions, we would need either a model trained specifically for HOI datasets or stronger filtering and post-processing on top of the VLM outputs.