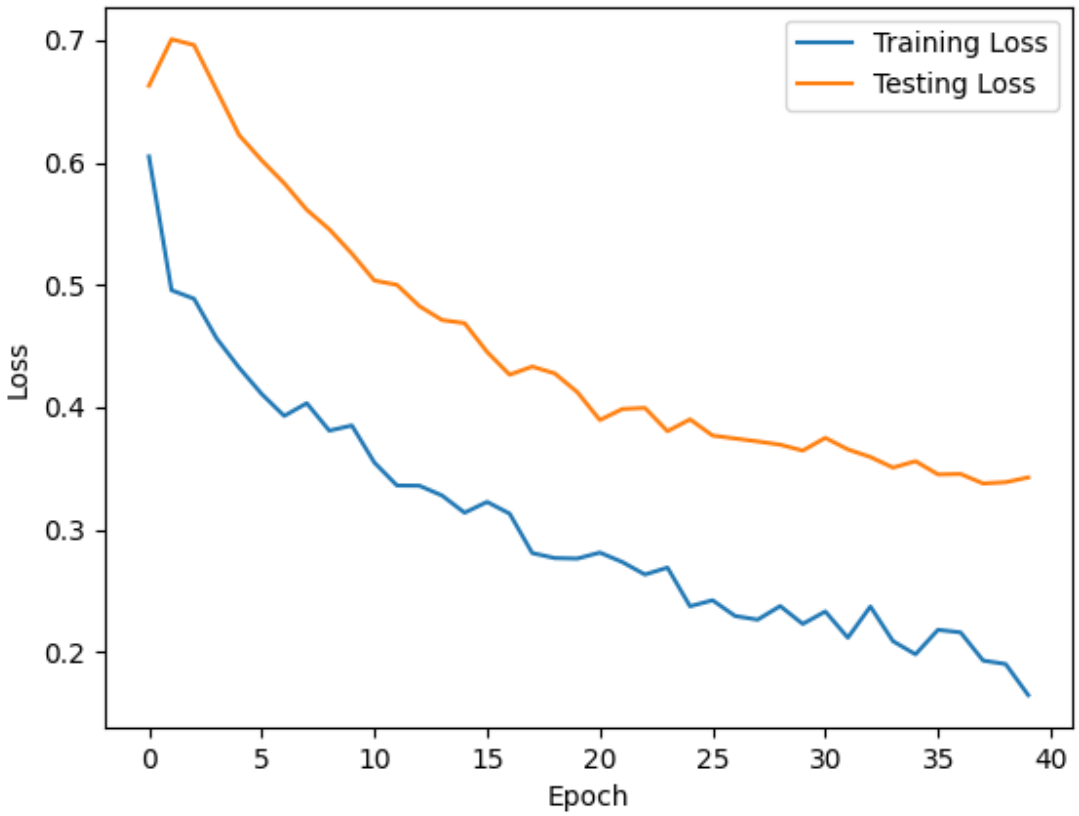
YODA – ELEC 475 Lab 4 Report

Noah Serhan

Step 3 – Model

I used the default PyTorch ResNet-18 model. After a while of our model producing results of around 67% accuracy, and us not being able to improve that, I made changes to the model and it now produces an accuracy of 86.25%, which I am happy with. Attatched below is our loss plot and confusion matrix:



Confusion Matrix Distribution

TP (61.25%)	FP (3.00%)
FN (10.75%)	TN (25.00%)

As we can see in the above confusion matrix, the majority of the allocation of the results (around 86%) are in the true negative and true positive sections, meaning the images that were classified correctly. Of that section, more correct predictions were positive (there was a car present) rather than correctly predicting that there was no car present. This Is because the model likely saw more images that had cars in them, then images that did not.

On the other hand, around 14% of the distribution was incorrectly classified. Of this, around 11% of them were falsely predicted negative, meaning the model predicted that there was no car in the image, when there was, and 3% went to the opposite. This is likely because it is easier for the model to miss a car that is present in an image, rather then see a car that is not at all present in an image, explaining this difference.

Step 4 – Method

Our mean IoU was 25127131319174395, so approximately 0.25. This mean IoU suggests that on average, the overlap between the predicted bounding boxes and the ground truth is around 25%, which generally is on the lower side when it comes to scoring an object detection model. Towards the finishing steps of our lab, we had trouble getting the final product to work, so I am happy with the result. Additionally, when we test out our model, it seems to work great. This low score could mean that while the model is detecting some aspects of the cars, it is not localizing them accurately, however our model does correctly identify any cars that are present.

Below are various examples of our functioning model:

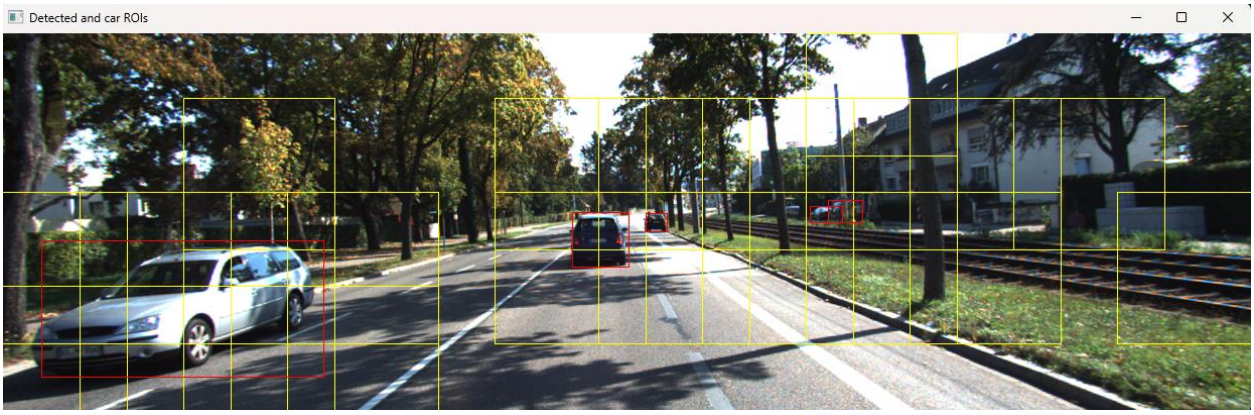


Figure 1: Picturing an example image with the ROIs and the cars within them in yellow and red boxes, respectively.

Figure 2: Picturing an example image with the ROIs and the cars within them in yellow and red boxes, respectively.

In *figure 1 and figure 2*, we can see the model accurately detected all the cars within the ROI's present in the image.

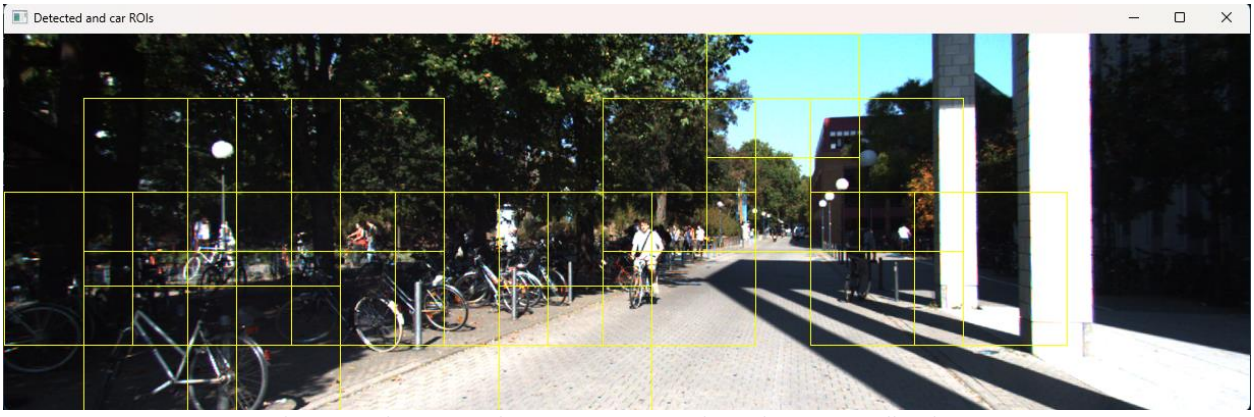


Figure 3: Picturing an example image, this time with no cars present and just the ROIs in yellow boxes.

In *figure 3*, we can see that there are no cars present in the image, and the model accurately did not detect any cars in the ROIs throughout the image.

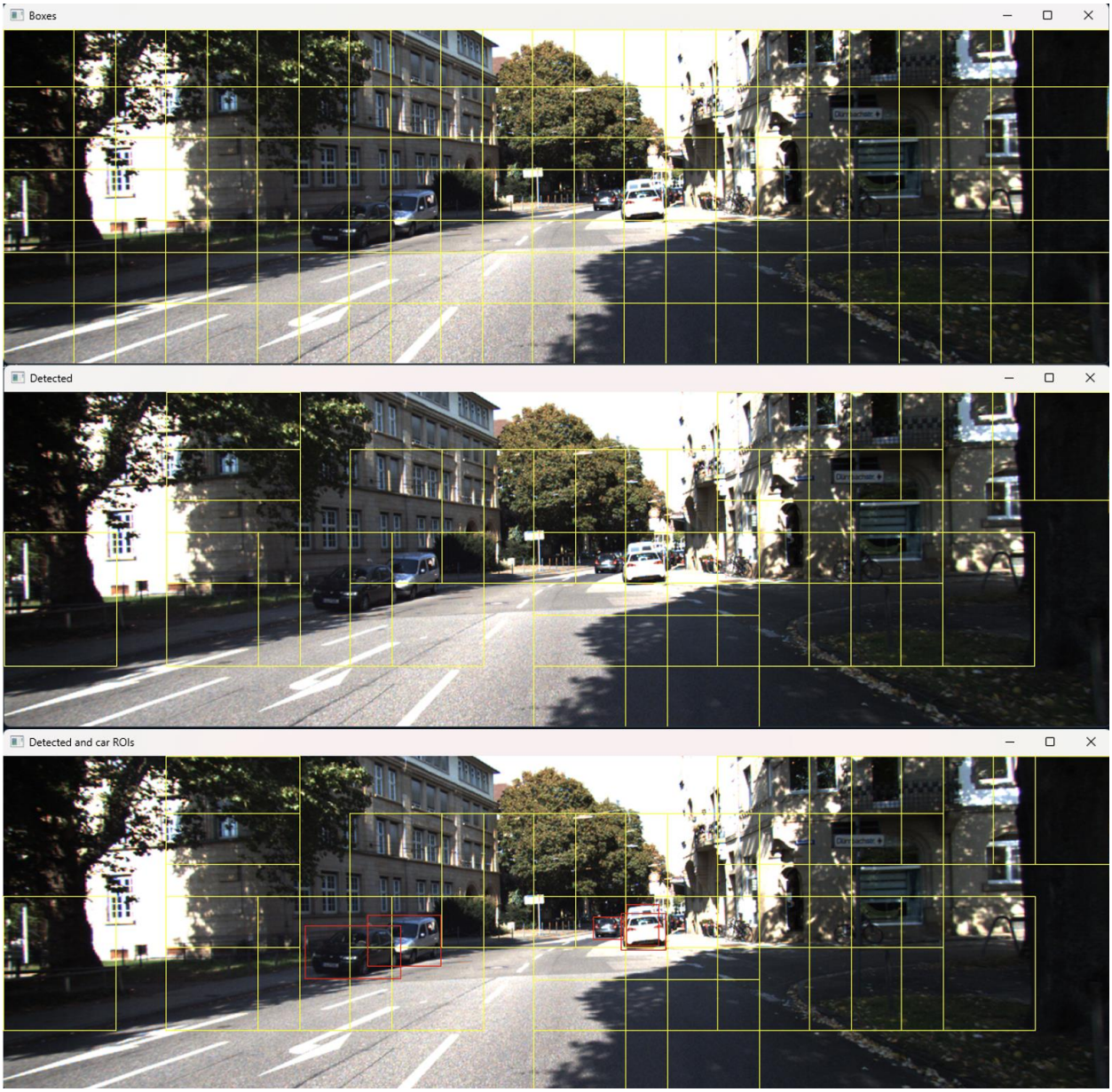


Figure 4

In *figure 4*, we can observe the step-by-step process that our model goes through, while detecting the objects in the image. The program firsts sets a grid size on the image to evenly distribute anchor points across the image. It then calls the "get_anchor_ROIs" method to generate the regions of interest and their corresponding bounding boxes based on the anchor centers found using "calc_anchor_centers". These boxes are then processed by the classification model, which detects whether or not the ROI contains a car and finally, the model outputs more precise red boxes, indicating the locations of these cars on the image.

Discussion

Our main challenge revolved around using the correct hyperparameter to bet fit our classification model. In the early stages of our model while using Adam, our loss plot values for

the testing would not drop, instead they would rise, while the training was values were progressing downwards, which immediately gave signs of overfitting. Many combinations of optimizers, schedulers and hyperparameters were tested, along with adding and removing certain layers from the model, until we eventually landed on using the SGD optimizer with the ReduceLROnPlateau scheduler. This change helped solved this issue of the validation loss continuing to rise, and our loss plot started looking much closer to the way we had hoped it would.

Additionally, as discussed earlier; our mean IoU was approximately 0.25. In most cases, we would aim for a mean IoU higher than this, as is especially the case for practical applications such as autonomous driving or precision agriculture, where accurate localization is critical and an IoU of 0.5 is often a threshold for considering a prediction to be correct. However, this mean IoU is much more acceptable than it seems for the specific requirements of this application. Some ways for improvement might be including more training data or performing more data augmentation, and examining the distribution of IoU scores across the dataset to see which specific instances the model performs particularly poorly or well, which would help us improve the model. After considering all this, our model did perform to the standard that we expected.