PROJECT REPORT

Project 2

Team 3

University *of*
New Haven

Introduction:

Computer vision is another area which has seen tremendous improvements in the recent past and object detection is one of the most popular use cases of that. Object detection enables not just enumeration of objects present in an image but also drawing bounding boxes around them for localization. YOLO(Object Detection) is one of the fastest and highly useful architecture for 'RealTime Object Detection' because it makes prediction in 'one forward pass of the network' only.

Here we trained a YOLOv8 model for the detection and localization of particular objects in the custom dataset. It contains images bearing class labels and added annotated bounding boxes. We hope by training the model on these images it will be able to use this to correctly predict, not only where objects are, but also their class in unseen images.

In this report the aim is to describe the methodology followed in training the model, the dataset was used for training and for experiments conducted and the results obtained. Next, we will examine the performance of the model (with respect to hyperparameters and training strategies) and explain why certain results from the previous experiment did not lead to the desired outcomes.

2. Method:

2.1 Network Architecture:

YOLOv8 is a contemporary object detection architecture that exploits YOLOv4 and YOLOv5 strengths. Speed, accuracy, and ease of use are all optimized when compared to the original. The network architecture is divided into three main parts:

Backbone: Feature extraction from input images are YOLOv8's backbone. It consists of series convolutional layers to identify some patterns and features. YOLOv8

backbone is based on a CSPDarknet, a feature map splitter reducing computation and improving efficiency of the model by processing it in parallel.

Neck: The neck is a place between the backbone and the head. Then it processes further the extracted features and refines them for object detection. PANet (Path Aggregation Network) in the neck is used in YOLOv8 to better aggregate feature across different layers of the model.

Head: Finally, the head of YOLOv8 model makes final predictions, including bounding boxes, class labels, objectness scores. Giving an image, the model outputs multiple predictions into multiple anchor boxes located in different spatial locations of the image, and these predictions are then further refined by nonmaximal suppression (NMS).

The model balanced performance and computational efficiency. The ability to do both classification and localization in a single pass through the network makes YOLOv8 very well suited for real time object detection applications.

2.2 Loss Function:

YOLOv8 uses a multi-part loss function that combines several loss components:

Localization Loss: This piece of the loss function reduces the error of predicted coordinates of bounding box. Most of the time it minimizes mean sqared error (MSE) or something similar to the ground truth bounding box coordinate.

Confidence Loss: What are YOLO models? YOLO models are meant to predict whether a bounding box will have an object. This confidence loss incurs a penalty on the case where predictions incorrectly score a box with high confidence, when the box does not contain any object.

Classification Loss: The next component of the loss is a measure of how well the model's predicted class labels are in agreement with the true class labels. Since typically from there, if it is a binary problem it will use binary cross entropy, if it is a categorical problem then use categorical cross entropy.

In addition to that, YOLOv8 also uses focal loss to solve class imbalance problem in object detection. Basically, focal loss is a loss function that gives more weight to unclear items than straightforward items.

Additionally, the combined loss function that the model learns from allows it to actually learn to classify the objects correctly, and also estimate its certainty of predictions.

3. Dataset:

3.1 Data Collection:

For this project we used manually annotated images from a variety of real world environments as the dataset. The images we have here are of different objects such as Road, building, tree etc. The images were annotated with bounding boxes around each object, and each object is assigned a class label.

a dataset that is highly diverse in terms of the scenes in it, including parks, urban streets, and indoor scenes, so the model gets to see a lot of different worlds. Annotations were performed using the LabelImg tool and the exported annotations is compatible with YOLO format.

Example image from the dataset:

3.2 Data Partitioning:

To assess the model's generalization ability, the dataset was split into three main subsets:

Training Set: In our experiments, we used 80% of the images to train the model. To improve diversity of the dataset and to prevent overfitting, these images were further augmented.

Validation Set: 10% of the images were set aside for validation. The model was monitored by the validation set to see how it performs during training and fine tunes the hyperparameters.

Testing Set: 10% of the images were set aside for testing.

A random split was used on the data and the training and validation sets were partitioned so that they represent the same parameterization of object classes.

3.3 Data Augmentation and Processing:

Data augmentation was applied to get more diverse training set and make strong model. These techniques include:

Mosaic Augmentation: Thus, this technique takes four images and combines them together into one image, which is more varied dataset. The model learns from various combinations of object contexts.
Random Scaling and Cropping: Randomly resizing and cropping images created variations of object size and position within the image.
Flipping and Rotation: On the other hand, we applied random horizontal flips and rotations so that the model is invariant to these transformations.

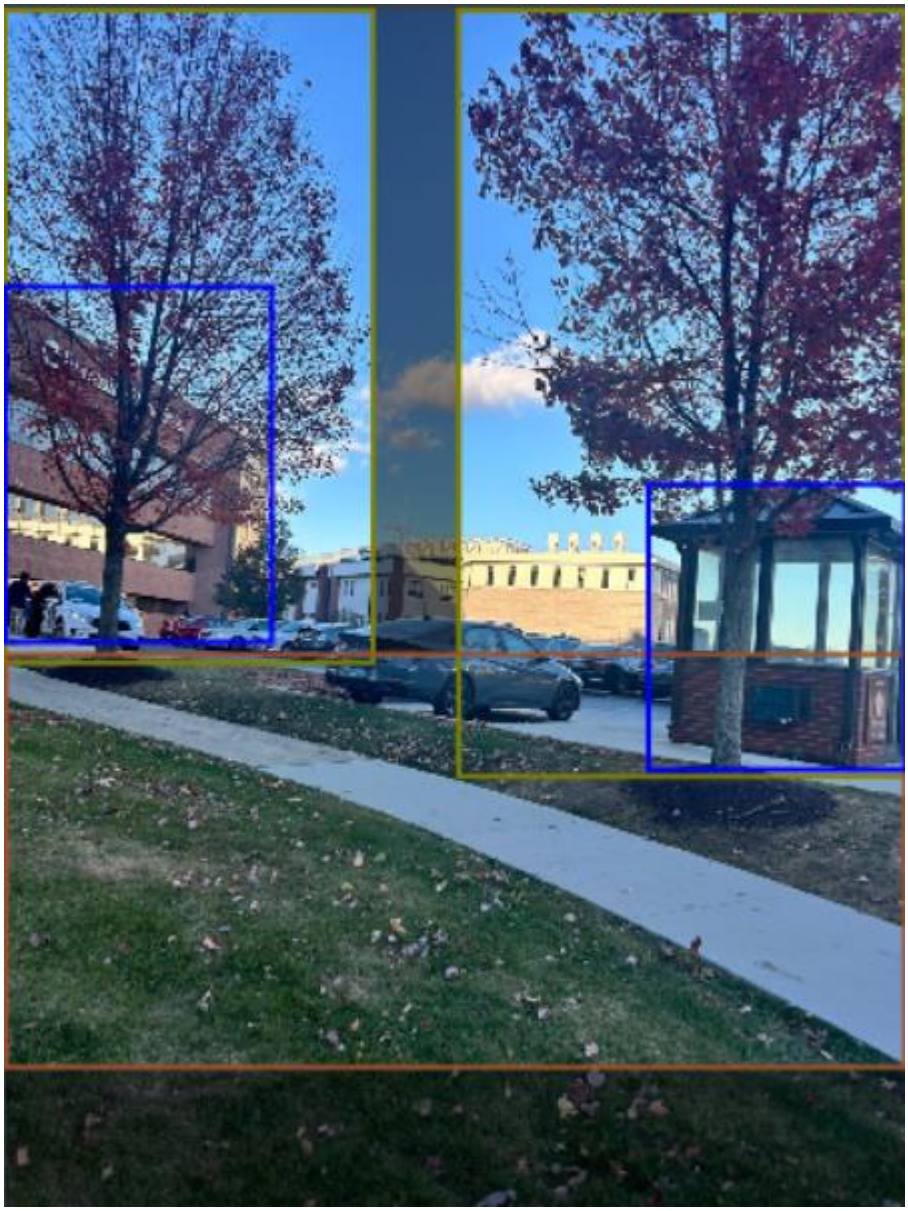
3.4 Normalization:

Image normalization is always necessary to make all the input images of the same scale. To make sure the model learns well from the images, I normalized the images to be within the range [0, 1] and standardized their RGB values using values of the mean and standard deviation for the dataset.

3.5 Sample Image and Annotation:

Here is an example from the dataset:

Image: A picture of a street, a building, a road and a tree.



4. Experiments and Results:

4.1 Hyperparameters and Best Parameters:

The training was performed carefully shifting hyperparameters in a trained model. Choosing hyperparameters was made based on the characteristics of the dataset as well as on the computational constraints. Some of the key hyperparameters used are:

Learning Rate (lr0): 0.1: A lower learning rate mean the model will converge faster. Learning Rate Scheduler (lrf): 0.This value represents the parameter value which can decay during training to allow learning rate to be diminished, making the model find a local minimum.

Momentum: 0.Momentum smooths out oscillation of gradient descent so that it helps accelerate gradient descent.

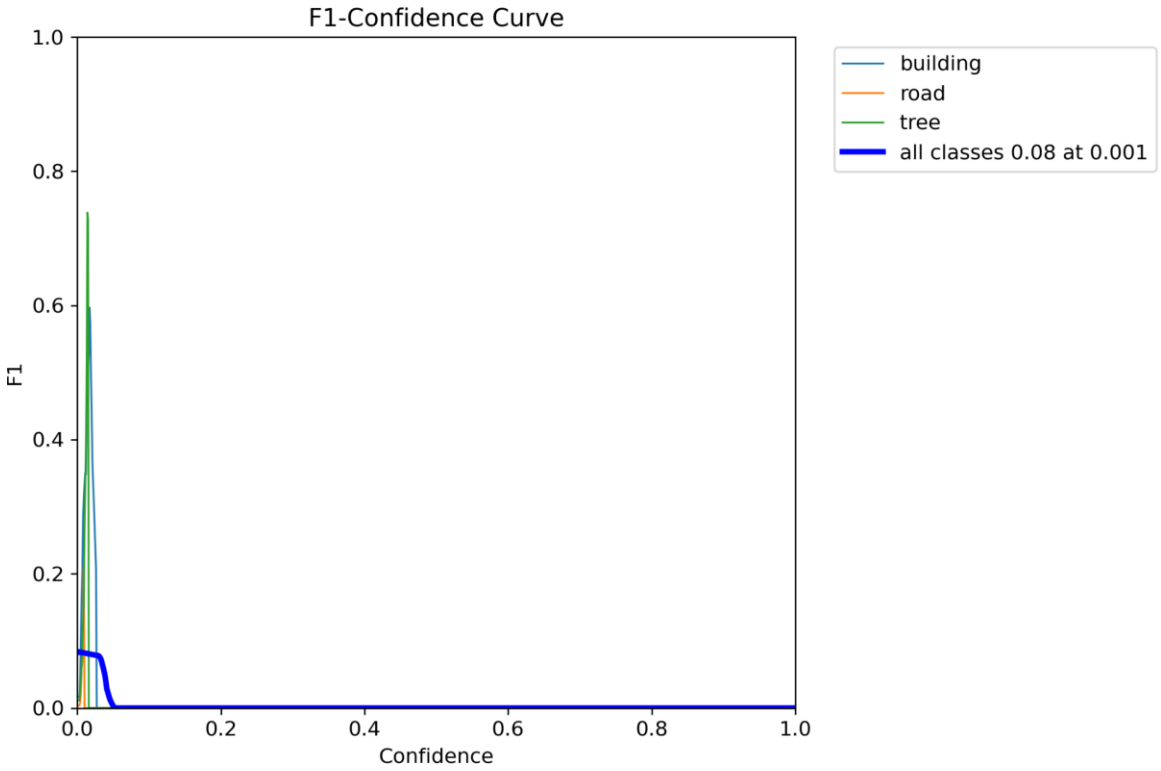
Weight Decay: 0.This regularization term helps prevent overfitting and it is by tagging it 0005.
Warmup Epochs: 1 – This keeps the initial learning rate warm, and slowly increases to avoid crazy large updates that could destabilize training.
Batch Size: When a larger batch size is used, training is more stable at the cost of more memory.
Mosaic Augmentation: 0.Degree of Mosaic Augmentation applied to training images. 5

4.2 Training and Validation Performance:

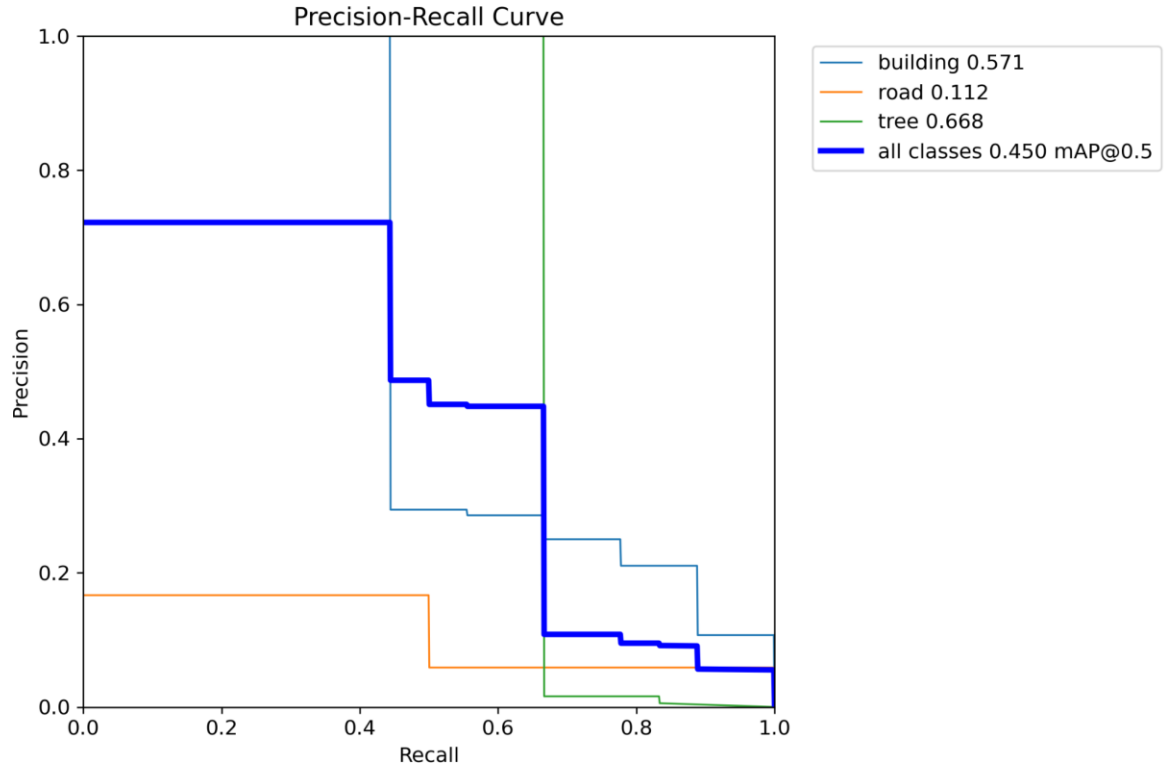
It was tracked and monitored how the model performs in training and validation across loss, mAP, and accuracy. The model converged fast in the first 106 epochs and training was very efficient.

The following plots summarize the performance:

F1:



P-R Curve:

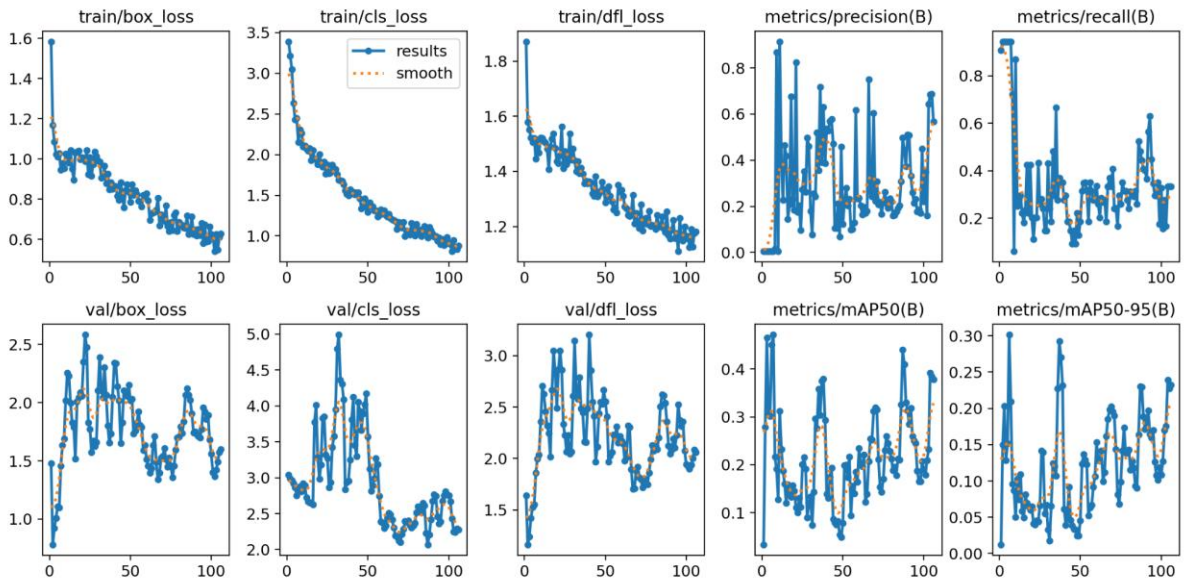


4.3 Pre and Post Training Comparison:

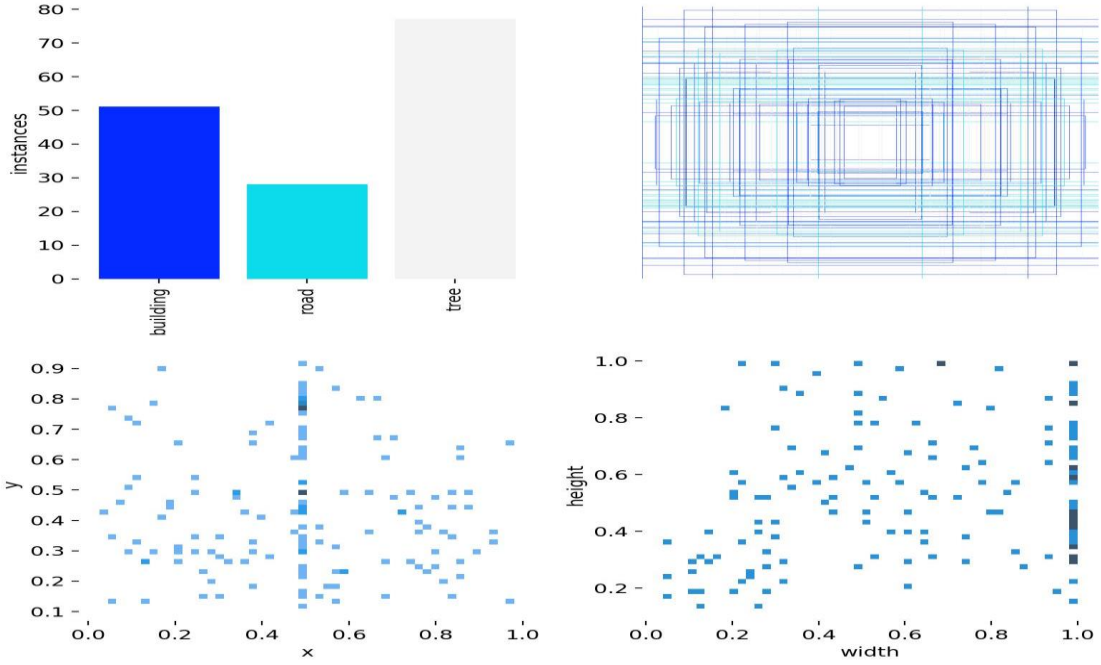
Pre-Training: The weight of the model was initialized randomly before training, and there were no accurate predictions. Though the loss at the beginning was very high, and mAP was close to zero, which means the model failed to detect the objects properly.

Post-Training: When I feed the model 250 epochs of training it sees amazing improvement in the localization and classification tasks. It was observed that passing through the higher IoU thresholds resulted in substantial gains in the mAP which confirms a better localizing capability of the model to the objects with high precision. In addition, the accuracy of the validation reached an excellent value and the model generalized very well to unseen data.

Final results


train/box_loss
train/cls_loss
results
smooth
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)

Labels


instances
building
road
tree
y
x
height
width

5. Conclusion:

Here we were able to train a of YOLOv8 model to objects in a custom dataset. In this dataset there are annotated images of an array of different object classes and data augmentation and normalization mean that the model generalises well to unknown or unseen data. The model was trained properly as evidenced by good accuracy scores and mAP, that proves it can not only detect but also localize objects.

There are future work to improve the model's performance such as, in further expanding the dataset, optimizing the hyper parameter further, and exploring of transfer learning to improve the model's performance in new regions. In addition, converting the model to be deployed in real time for applications such as autonomous vehicles or surveillance systems is a very promising next step.