Assignment 1: Object Detection using YOLOv8

Abdul Bhutta
Electrical and Computer Engineering
Toronto Metropolitan University (TMU)
Toronto, Canada
abdul.bhutta@torontomu.ca

I. Introduction

Object classification, a crucial task in computer vision, involves identifying the class of an object within an image. This is further enhanced by object localization, which classifies and pinpoints the object in the image using bounding boxes. This study delves into the cutting-edge YOLO (You Only Look Once) algorithm, a state-of-the-art object detection model. It inspects how this algorithm is leveraged to train a custom dataset, underscoring its significance in the field. Object detection is necessary in many real-world applications, such as autonomous vehicles, surveillance, and medical imaging. It is a challenging task that requires the model to detect and classify multiple objects in an image while also providing the bounding box coordinates for each object and extending further to object tracking, as shown in Figure 1. It requires consideration of various factors to detect objects, and some challenges, such as occlusion, scale variation, and lighting conditions, can degrade the performance of object detection algorithms.

Fig. 1. Object Detection with Bounding Boxes

In the context of object detection, various evaluation metrics come into play. These metrics, including precision, recall, and mean average precision (mAP) at different intersections over union (IoU) thresholds, such as 0.5 or 0.5 to 0.95, are instrumental in assessing the model's performance. They provide a quantitative measure of how well the model can detect and classify objects, giving us a clear picture of its effectiveness in real-world scenarios. The precision calculates the accuracy of how many objects were correctly predicted or correct positive predictions, and recall calculates how many actual objects were correctly predicted or how many were detected by the model out of all actual objects. The equation to calculate each metric is shown in equations 1 and 2, respectively.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \quad (1)$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \quad (2)$$

Intersection over Union (IoU) is a critical concept in object detection. It measures the overlap between the predicted and ground truth bounding boxes, quantitatively measuring the model's accuracy. In simpler terms, it tells us how much the predicted box matches the actual box. The IoU is calculated using the equation below.

$$IoU = \frac{AreaofOverlap}{AreaofUnion} \quad (3)$$

Mean Average Precision (mAP) is a metric used to summarize the precision-recall curve for all the classes in the dataset and is the mean of the average precision for each class. When the mAP at an IoU threshold of 0.50 is used to evaluate the model performance, it combines the precision and recall of all the classes in the dataset. The threshold of 0.5 allows the model to detect objects that have an overlap of 50% or more with the actual bounding box. This leads the model to be more lenient and not too strict and is one of the most widely used benchmarks for object detection models. mAP at 0.5 to 0.95 is an alternative metric for evaluating model performance at different IoU thresholds. It is a stricter benchmark than mAP at 0.5 and increments the IoU threshold by 0.05 from 0.5 until 0.95. It is a more severe evaluation method and requires the model to have a higher accuracy through the various IoU thresholds and produce more robust bounding boxes while also being able to classify the objects correctly. This metric is used for real-world applications where the localization of the object is critical and requires a higher level of accuracy.

In this assignment, the YOLOv8 algorithm was used to train a custom dataset consisting of various traffic signs. The dataset was divided into training, validation, and testing datasets, while the model was trained using the training dataset and evaluated using the validation dataset during training. The model's performance was assessed using various evaluation metrics, such as precision, recall, and mAP at different IoU thresholds. Lastly, the trained model was evaluated using the test dataset to assess its performance and accuracy on unseen data. Furthermore, the results were analyzed to determine how

the algorithm produced the final output and visualized various stages in the algorithm using an inference test on a single image.

II. YOLO Algorithm

The YOLO algorithm is a single-shot detector that can detect multiple objects in an image and create bounding boxes simultaneously using a single forward pass of the network, allowing it to be used for real-time object detection. It uses a grid-based approach to divide the image into a grid, and each grid is responsible for predicting the bounding boxes and the class of the object while also providing a confidence score for each bounding box. YOLO algorithms have evolved over time, with each version building upon the previous one. The most recent version by Ultralytics, YOLOv8, stands out with its unique features. It offers five versions: nano, small, medium, large, and extra-large, each with varying layers and parameters. The layers and parameters are increased as you go higher up in the models, leading the higher models to be more complex and requiring more training time but can potentially deliver better accuracy.

In this assignment, YOLOv8s was implemented, a model that requires more training time than the nano version. However, given the small dataset, this model can still train effectively and provide good accuracy. YOLOv8s has a few changes to its architecture and introduces two significant improvements: a new backbone network, CSPDarknet, which is more efficient and faster than its predecessor, and a new activation function, SiLu (Sigmoid Linear Unit), which replaces the ReLU activation function and accelerates the learning process. Furthermore, it has changed to anchor-free detection, which leads to less training time.

The YOLO model has three main components: Backbone, Neck, and Head, which oversee specific tasks in the neural network. The backbone network extracts various features, such as shapes or objects, from the image using a pre-trained model (CSPDarknet). It creates multiple feature maps from each layer to produce features at low and high levels of abstraction to capture the essential information from the image. The neck module combines the features extracted from the backbone module from the different layers to help detect objects at various scales. It contains a feature pyramid network (FPN), path aggregation network (PAN), and spatial pyramid pooling (SPP) modules that are used to combine the features from the backbone network and create multiple new scale feature maps that can be used to detect objects in the image. The head takes the new feature maps constructed in the neck module, which consists of convolutional layers and detection layers to predict the bounding boxes, class of the object, and the objectness score to produce a final output. Each feature map grid cell produces bounding boxes containing an objectness score and the class probabilities.

A. Loss Function

The loss function used in the current YOLOv8 training model combines three different loss functions box_loss, cls_loss, and dfl_loss. The box_loss calculates the bounding box regression loss, which measures how close the predicted bounding box is to the ground truth bounding box. The cls_loss or classification loss is used to calculate the classification loss, which measures how well the model can classify the object in the image and uses the standard Binary Cross Entropy (BCE) loss function. The dfl_loss or detection feature loss is used to calculate the distribution feature loss, which measures the distribution of the predicted bounding box coordinates and is used to improve the localization of the object in the image. The total loss is calculated as the weighted sum of the three loss functions, as shown in the equation below.

$$TotalLoss = w_{box} \times box\_loss + w_{cls} \times cls\_loss + w_{dfl} \times dfl\_loss \quad (4)$$

where w_{box}, w_{cls}, and w_{dfl} are the hyperparameters to control the weights for the box, cls, and dfl loss functions, respectively. The weights are used to balance the contribution of each loss function to the total loss and improve the model's performance. During training, all weights and parameters of the loss functions are set to the default values.

III. Training, Validation, and Testing

The Dataset found on Kaggle contains a training dataset consisting of 3530 images and a validation dataset of 801 images. The algorithm applied various preprocessing techniques to the image before training the model. All the images were resized to 640 by 640 pixels, and multiple data augmentation techniques were used, such as blur, grayscale, and contrast-limited adaptive histogram equalization (CLAHE), to improve the image's contrast. The annotations for each image were saved in a .txt file and YOLO format, where each line contained the class of the object, the bounding box coordinates, and the image's width and height.

The algorithm was used to train the model for 100 epochs with a default batch size of 16. The weights were initialized using the yolov8s.pt pre-trained model. It was trained using the Adam optimizer, and a learning rate of 0.001 was initially applied, demonstrating the model's adaptability. However, with its auto-learning rate feature, the algorithm dynamically adjusted the learning rate, starting from 0.000526 with a momentum of 0.9. An early stopping technique was also implemented, allowing the model to halt training when the validation loss did not improve for ten epochs, thereby preventing overfitting and conserving computational resources. The best weights were saved to a file and used to evaluate the model on the test dataset. The model was trained using Google Colab Pro, which provided access to a Tesla T4 GPU with 16GB of memory and took approximately 2.2 hours for training.

IV. Results

A. Training and Validation Results

Fig. 2. Training Batch 0

The algorithm applied various data augmentation techniques to the image before training, and images used for batch 0 for training are shown in Figure 2. The model was trained for 84 epochs using the train and validation datasets. At the same time, the best results were obtained at epoch 74 for training, as shown in Table 1, where after ten epochs, the model did not improve and resulted in stopping the training process. The model achieved a box loss of 0.503, cls loss of 0.518, and dfl loss of 0.948. It also achieved a precision of 0.951 and a recall of 0.901. As the epoch increases, the values of the box_loss, cls_loss, and dfl_loss decrease while the precision and recall increase, which shows the model is learning and improving as the number of epochs increases. Each metric graph is shown in Figure 3.

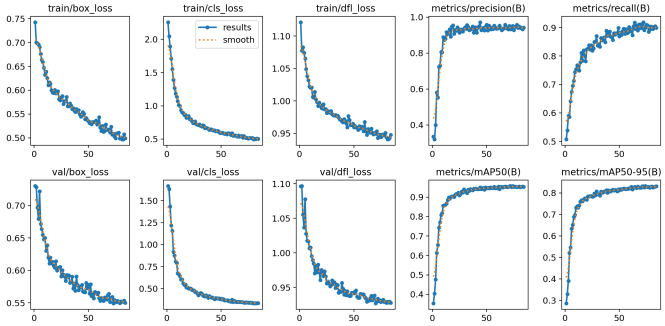

Fig. 3. Metrics Results vs Epochs

TABLE I
Training Results

Epoch	box_loss	cls_loss	dfl_loss	precision(B)	recall(B)
1	0.74284	2.2586	1.1215	0.33474	0.50679
20	0.59337	0.84507	0.98938	0.90672	0.82416
40	0.55516	0.642	0.96744	0.94331	0.88108
60	0.54337	0.58628	0.96791	0.9362	0.88755
74	**0.503**	**0.51815**	**0.94805**	**0.95126**	**0.90061**
80	0.50098	0.49456	0.94386	0.94784	0.90039
84	0.49907	0.5054	0.94785	0.94112	0.89888

The model's validation box loss was 0.554, cls loss was 0.340, and dfl loss was 0.931, as detailed in Table 2. It demonstrated a precision of 0.906 and a recall of 0.911, further affirming its accuracy. As the table below shows, the model's training results are great, and its peak performance was achieved at epoch 74, a testament to its robust capabilities and untapped potential.

TABLE II
Validation Results

Epoch	val/box_loss	val/cls_loss	val/dfl_loss
1	0.73031	1.6673	1.0964
20	0.61407	0.50294	0.97536
40	0.57777	0.40075	0.94815
60	0.55748	0.37218	0.93627
74	**0.5542**	**0.34056**	**0.9314**
80	0.55228	0.3331	0.92788
84	0.54986	0.33618	0.92765

The confusion matrix for the training data is shown in Figure 4, where the values have been normalized, meaning the values are between 0 and 1 and can be considered as probabilities where the speed limit of 100 signs had 96% accuracy of being predicted correctly.

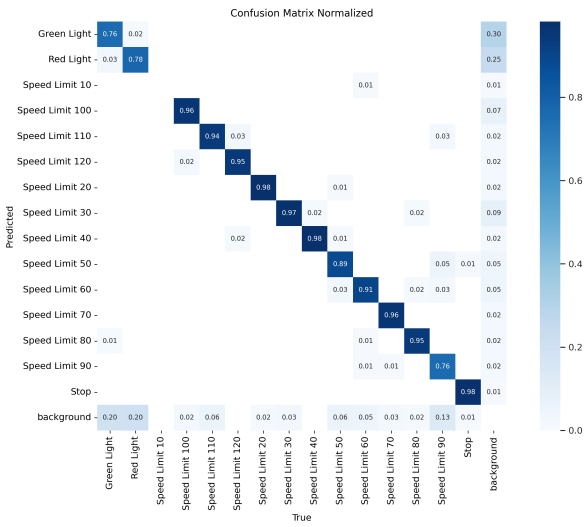

Fig. 4. Confusion Matrix

B. Test Dataset Results

Once the model was finished training, the best weights from epoch 74 were saved and used to evaluate the model on unseen data using the test dataset. For the test dataset, the model achieved a precision of 0.906, recall of 0.91, mAP50 of 0.949, and mAP50-95 of 0.81. The model performed well on the test dataset and achieved high precision and recall for most classes, as shown in Table 3.

TABLE III
TEST RESULTS

Class	Instances	Precision	Recall	mAP50	mAP50-95
Green Light	110	0.885	0.837	0.908	0.516
Red Light	94	0.793	0.733	0.756	0.49
Stop	50	0.972	1	0.995	0.926
Speed Limit 10	3	1	0.854	0.995	0.759
Speed Limit 20	46	0.919	0.978	0.976	0.899
Speed Limit 30	60	0.823	0.933	0.968	0.881
Speed Limit 40	53	0.928	0.962	0.968	0.875
Speed Limit 50	50	0.866	0.9	0.943	0.861
Speed Limit 60	45	0.977	0.932	0.968	0.861
Speed Limit 70	53	0.922	0.887	0.958	0.863
Speed Limit 80	61	0.931	1	0.991	0.874
Speed Limit 90	34	0.883	0.891	0.955	0.797
Speed Limit 100	46	0.911	0.978	0.985	0.862
Speed Limit 110	21	0.871	0.81	0.905	0.82
Speed Limit 120	44	0.906	0.977	0.965	0.874
All	770	0.906	0.911	0.949	0.81

C. Inference Results

Fig. 5. Inference Test Image

An inference test was conducted on a randomly selected image, Figure 5, from the test dataset to verify and analyze how the model executes on a single image. The image goes through 23 different processing stages before the final output is generated. The image is first passed through stages 0 to 9 while generating multiple feature maps, and stages 4,6 and 9 feature maps are sent to the Neck module. The last eight feature maps from the three stages (4,6, and 9) are shown in Figures 6 to 8. Each feature map allows the model to capture different features or unique characteristics in the image, such as edges, corners, and textures.

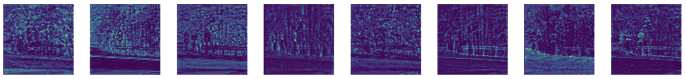

Fig. 6. Stage 4

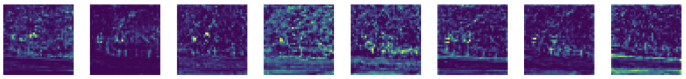

Fig. 7. Stage 6

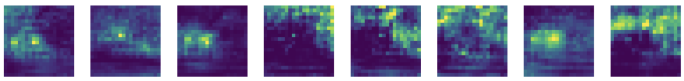

Fig. 8. Stage 9

The feature maps are then passed to the neck layer and used to concatenate the features from the different scales to create new feature maps in layers 15, 18, and 21, as shown in Figures 9 to 11. These three layers are passed to the head layer to predict the bounding boxes, class of the object, and the objectness score to generate a final output, Figure 12.

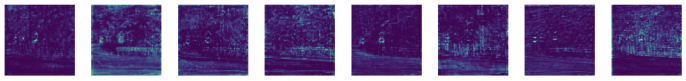

Fig. 9. Stage 15

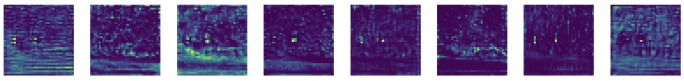

Fig. 10. Stage 18

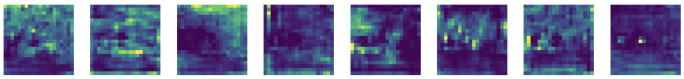

Fig. 11. Stage 21

Fig. 12. Inference Test Image with Predicted Bounding Boxes

REFERENCES

[1] D. Reis, J. Kupec, J. Hong, and A. Daoudi, "Real-time flying object detection with yolov8," arXiv.org, https://arxiv.org/abs/2305.09972 (accessed May 13, 2024).

[2] J. Solawetz, "What is Yolov8? the ultimate guide. [2024]," Roboflow Blog, https://blog.roboflow.com/whats-new-in-yolov8/ (accessed May 15, 2024).

[3] A. Acharya, "Yolo Object Detection explained: Evolution, algorithm, and applications," Encord, https://encord.com/blog/yolo-object-detection-guide/ (accessed May 16, 2024).

[4] D. & A. Solutions, "Losses and their weights in yolov8," LinkedIn, https://www.linkedin.com/pulse/losses-weights-yolov8-dsaisolutions-x1ggf#:~:text=œbox%20refers%20to%20the%20loss,7.5%2C%200.5%2C%20and%201.5., (accessed May 16, 2024).