2D Object Detection on Waymo Open Dataset

Liam Adams (lbadams2) Komal Kangutkar (kmkangut)

***Abstract*—One of the most important tasks for an autonomous driving system to perform is object detection. Using cameras, objects can be detected as 2 dimensional features within an image. Deep neural networks are well suited for this task as they are able to process massive amounts of image data. Once trained the network can be deployed to the system on the car, and it can make inferences on objects in real time. Our goal is to identify the types and dimensions of objects a self driving car would encounter.**

We performed 2D object detection on the Waymo Open Dataset [1] using the YOLO Darknet Architecture [2]. We implemented the architecture using Tensorflow, and transformed the Waymo dataset to fit the network. The architecture is a deep convolutional neural network, consisting of a series of 2D convolution and max pooling layers with alternating filter sizes. We used the loss function described in the original YOLO paper [3]. In order to detect multiple overlapping objects we made use of anchors [3], and to select the best objects during inference we used the non max suppression algorithm.

We evaluated our model using precision and recall. We identified a true positive detection with the correct class and exceeding a certain overlap threshold with the ground truth object. We achieved a precision of .037 and a recall of .037. We also calculated mean average precision, which is the area under the precision-recall curve. We achieved a mean average precision of 18.

I. Model Training and Selection

A. The Model

2D object detection models are normally either single stage or double stage models [4]. The multiple stage models train 2 networks: one for proposing regions of interest and one for identifying objects in those regions. The single stage models divide the image into a fixed set of regions or boxes, eliminating the region proposal network. Due to our limited computational resources we will use the single stage method. We draw inspiration from YOLO [3].

YOLO stands for *You Only Look Once*. It was named in order to highlight the contrast between it and other object detection systems. YOLO considers each image in its entirety, rather than using a sliding window approach. It frames object detection as a regression problem with a fixed set of parameters per image. The image is first divided an S x S grid with each grid cell having a fixed number of anchors (B) that can make object detection predictions. By making use of anchors, YOLO is able to detect a varying number of objects of different sizes in an image. Each anchor makes a prediction of a bounding box which has 5 components: $(t_x, t_y, t_w, t_h, confidence)$. The center coordinates t_x, t_y are predicted relative to the grid cell location (c_x, c_y). t_w and t_h are in the log scale and relative to the anchor dimensions (p_w, p_h). To get the actual bounding box dimensions [5], we do:

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h}
\end{aligned}
$$

The prediction also includes softmax over C classes. Thus, the output from the network is a S x S x B x (5 + C) tensor. For our network, we used 5 anchors per grid cell with the image divided into a 13 x 13 grid. There were 5 classes of objects *viz.* Vehicle, Pedestrian, Cyclist, Sign and Unknown. So the prediction from the network was a 13 x 13 x 5 x 10 tensor. See Figure 1.

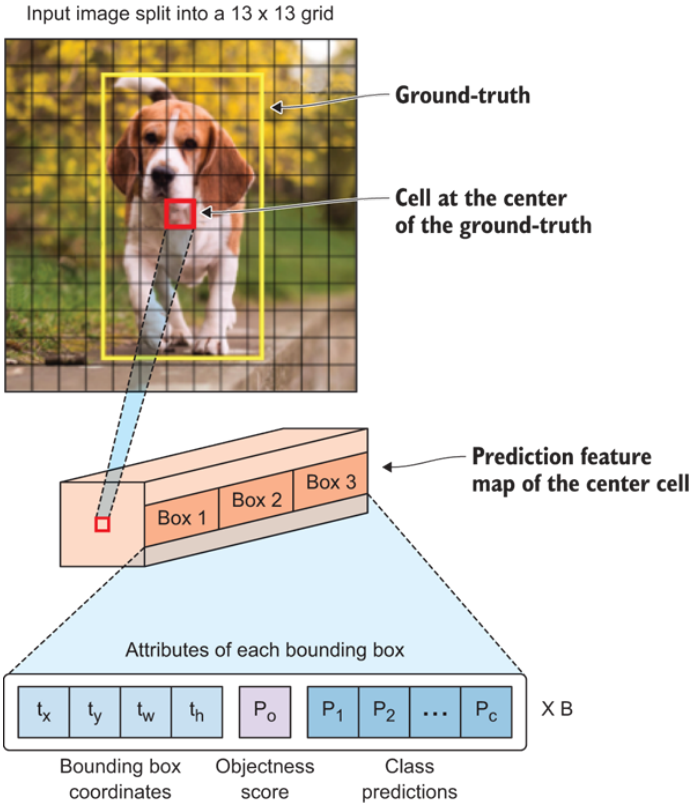

Fig. 1. Prediction vector [6]

We implemented the Darknet architecture [2] made available by the YOLO authors using Tensorflow [7]. This architecture is a deep convolutional neural network with alternating filter sizes. As the network progresses the depth gradually increases. Batch normalization and max pooling are employed through-

out the network. The activation function used is Leaky Relu. The network generates a 4D output volume corresponding to the grid, anchors, and prediction values we are interested in: 13 x 13 x 5 x 10. Figure 2 shows an example excerpt from the network. In between each layer there is max pooling to reduce dimensionality. There are also max pooling layers of stride 1 inserted, so the dimensionality does not reduce too rapidly.

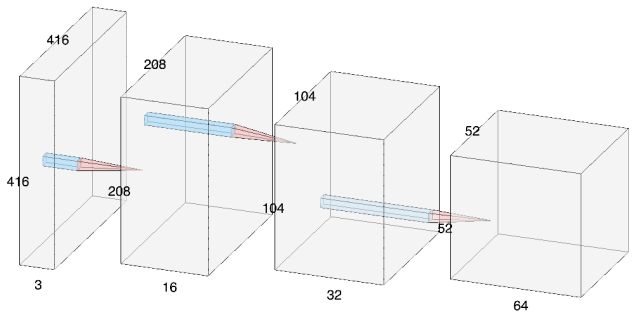

Fig. 2. Network Excerpt

The loss function used is the same as [3]. There are 3 component losses calculated and summed for the total loss: the coordinate loss, the confidence loss and the classification loss.

Non-Max Suppression (NMS) is used to keep only the bounding boxes with high confidence in the final prediction. Figure 3 shows the NMS process.

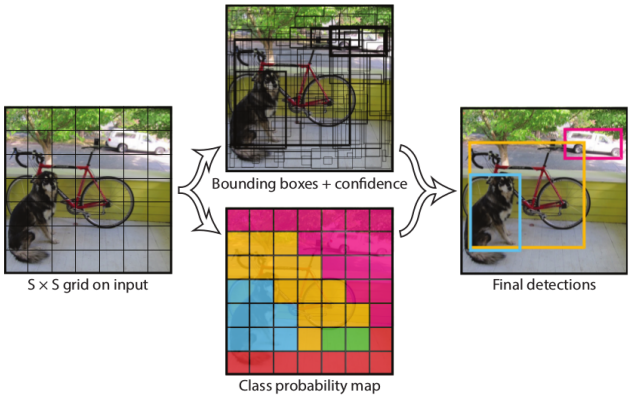

Fig. 3. Non-Max Suppression [3]

B. Baseline

We are comparing our results against the original YOLO results [3]. The original YOLO was tested against the VOC 2012 test set. We will use the same network architecture and loss function as described in the paper [3]. However for the training and test set we will be using the Waymo Open Dataset [1]. The original YOLO paper reported results as Mean Average Precision (mAP) and F1 Score. We will also use these metrics and additionally report localization error. The original authors obtained a mAP of 57.9 and F1 score of .59. In order for us to calculate mAP we will need several data points of precision and recall. We will use the precision and recall scores from each epoch on our validation set to generate a precision-recall curve. We then calculate the area under that curve for mAP.

We will diverge from the original model in the weights used in the loss function and our method for transforming coordinates. The original YOLO predicted the the square root of the width height coordinate, we will be predicting the natural log of the width height coordinates, and using the exponential to recover the true width and height. This keeps the predictions and thus the loss smaller, which should prevent exploding gradients. Another difference in our approach is that we will be resizing the images to 416 x 416 and transforming the true coordinates appropriately before feeding them to the network.

II. Experimental Section

A. Metrics

In order to determine if a predicted box overlaps with a ground truth box we use Intersection over Union, which is the area of the intersection of the boxes divided by the area of the union of the boxes. We define an IOU threshold and an object confidence threshold. In order to generate candidates for comparison to the ground truth objects, the detections from our network must first pass the object confidence threshold. Those detections' bounding boxes will then go through NMS to generate non-overlapping candidates. Now we calculate the IOU between these candidates and the ground truth boxes. If a prediction predicts the correct class and surpasses our IOU threshold, that counts as a true positive.

From this we can calculate the precision and recall. We also collect another metric called the localization error. This includes detections that passed the object confidence threshold, NMS, and correctly predicted the class of the ground truth, but did not surpass the IOU threshold. If we plot our precision and recall against one another with precision on the y-axis and recall on the x-axis we can calculate the area under that curve to get Mean Average Precision (mAP). We plotted our precision recall curve for our validation set and calcuated the mAP over these points.

B. Model Selection

The hyperparameters specific to our task are grid stride, scaled image width, scaled image height, object confidence, NMS threshold, IOU threshold, number of anchors, coordinate loss weight, object loss weight, no object loss weight, and classification loss weight.

We first had to scale the images from 1920 x 1280 (width x height) down to a smaller square image. This downscaling to a square image results in some distortion due to the different aspect ratio, but we wanted our coordinate transformations during training to happen on the same scale for both dimensions. We chose a scaled width and scaled height of 416 x

416 as that is a common size for images passed to deep convolutional networks. We then chose a grid stride of 32 to give a 13 x 13 grid. We thought 13 x 13 was a good compromise between granularity (and more network weights to learn) and coarseness. We chose 5 anchors which means we can potentially predict 5 objects per grid cell. More than 5 objects per cell is unlikely given the size of the objects we're looking for, and our relatively close distance to them. Also 5 allows us to use a diverse set of shapes for our anchors and hence the dimensions of the objects we are trying to detect. Each anchor box has different dimensions, and we felt that more than 5 would be redundant while less than 5 would not have captured a diverse enough set of objects.

As for the various threshold parameters, we chose an object confidence of .6. For a prediction, the object confidence is calculated by multiplying the original object confidence from the prediction vector by the maximum class probability. We did try higher object confidences but were not getting enough predictions. We also felt that .5 would be too low as that would basically be a coin flip. .6 was a good medium. We chose a NMS threshold of .5. Non-Max Suppression takes the predictions that have just passed the object confidence threshold, orders them by confidence and processes them from greatest to least. When it encounters a prediction with an IOU above .5 with a previous object with higher confidence, it discards the one with lower confidence. We chose .5 as this is the default for the Tensorflow [7] Non-Max Suppression algorithm. The final IOU threshold is between the predictions and ground truths, and is used to distinguish a detection from a non-detection. We chose a value of .5 here, we tried higher thresholds but that was resulting in too few detections, which is detrimental to the loss function.

We chose a coordinate loss weight of 1, an object loss weight of 5, a no object loss weight of 1, and a classification loss weight of 1. We weighed the object loss higher because the most important part is to detect the object first, then determine the coordinates. The object loss weight is used in the sum squared error between the ground truth objects and the corresponding positions in the grid of the prediction tensor. The no object loss weight is used in the sum squared error between the absence of objects in the true grid and the corresponding positions in the grid of the prediction tensor. It is important to detect objects when they are present and also to not predict objects that are not present. The coordinate loss weight is used on the sum squared error between the true coordinates and their respective prediction coordinates, only on those positions of the true grid containing an object. The classification loss was also only used on the positions of the true detections. Our motivation on these loss weights was mainly to weigh the object detection higher than all other predictions.

As for the network itself, we used an L2 regularizer with a regularization factor of 5e-4 and Adam optimizer with the learning rate set to 1e-4. The regularization factor was chosen in line with [5]. For the learning rate, we first tried 1e-3, which is the default for Adam, but our network was not learning. After reducing to 1e-4 the loss began to gradually decrease. We also chose a batch size of 32 and we trained the network for 15 epochs. Fig. 4 shows the training and validation loss curves.

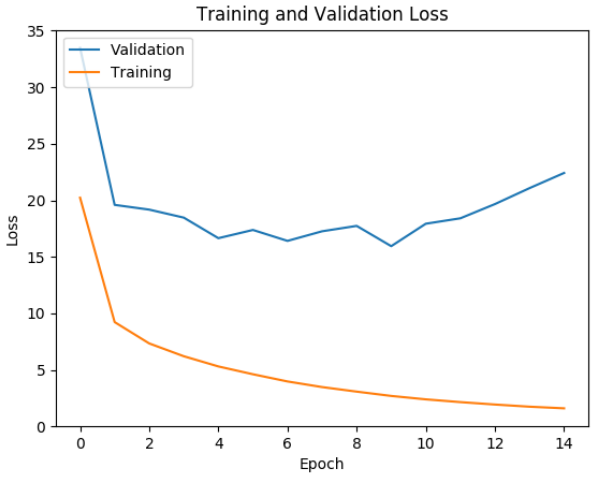

Fig. 4. Training and Validation loss

C. Performance and Comparison to Baseline

We achieved a mAP of 18 on the validation set, and an F1 score of .148 on the test set. The YOLO baseline [3] achieved a mAP of 57.9 and F1 score of .59. See Table I. Below we've included some example predictions of our network on the test set. You can see that it detects most objects, however the boxes are not always the appropriate size. Also there are some stray overlapping boxes that weren't filtered out. It is even able to make decent predictions in adverse weather conditions, as you can see Fig. 8.

We measured a localization error of 17%, which means 17% of our predictions were on a true object, however the bounding box of the prediction did not meet the IOU threshold. You can see a possible example of that in Fig. 5 of the single car. We included predictions having an IOU between .1 and .5 with a true object in the localization error. Perhaps we did not penalize the coordinate loss enough. Our object loss was 5 times that of our coordinate loss, a future improvement could be to even this ratio out some.

In the images from the test set, a red box indicates a car and a green box indicates a pedestrian. You can see in Fig. 7 a misclassified pedestrian. This was a problem for the entire test set. It generally failed to find any pedestrians, and the pedestrian classifications it did make were incorrect, as you can see with the trash can Fig. 7. However the objects classified as pedestrians were generally of the right shape a pedestrian would be. This indicates that one of our anchor boxes is influencing its corresponding prediction vector to learn objects in the shape of pedestrians.

The data overall was very imbalanced, with many examples examples of vehicles and very few pedestrians. We did not

Fig. 5. Single car

Fig. 7. Misclassification of trash can as a pedestrian

Fig. 6. Multiple cars

Fig. 8. Rainy weather

account for this in our model and if we had to do it again this would definitely be an area of improvement. Our anchor boxes did allow us to learn overlapping objects, and objects of different shapes, but we just did not provide enough data for the anchors that would correspond to other objects.

The Waymo data was very repetitive and sequential in nature as the images were taken from short time periods by multiple cameras in the same scene. The data is like this so other machine learning tasks in motion tracking could be performed. We may have achieved better performance on a dataset that was not so sequentially correlated. We did shuffle the training dataset in order to break up these correlations so our network

did not learn unintended features. Also perhaps 15,000 images was not quite enough, especially with the class imbalance.

Another potential problem is with our resizing of the image. We resized 1920 x 1280 images to 416 x 416 images, in the process changing the aspect ratio. This causes some distortions which are visible in the images shown. The coordinates of the bounding boxes on the true objects were transformed to take this resizing into account, so when calculating the loss we were comparing coordinates in the same domain. However it would be interesting to train the network on non-distorted images and measure its performance.

TABLE I
RESULTS

	Baseline	Results
mAP (Validation mAP)	57.9	18
F1 Score	.59	.148

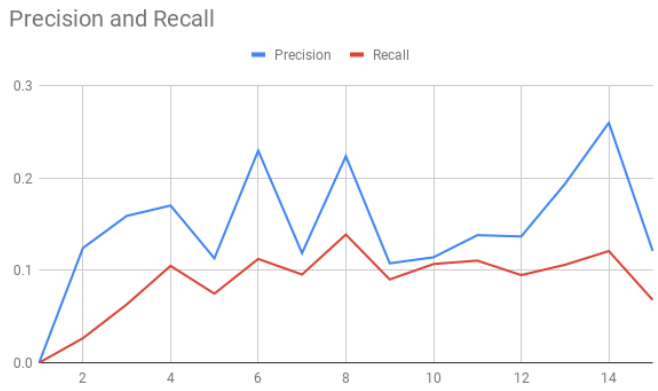

Fig. 9.

REFERENCES

[1] Waymo. ((2019)) Waymo open dataset. [Online]. Available: https://waymo.com/open

[2] J. Redmon. ((2019)) Darknet yolo. [Online]. Available: https://pjreddie.com/darknet/yolo/

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," 2015.

[5] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[6] [Online]. Available: https://livebook.manning.com/book/grokking-deep-learning-for-computer-vision/chapter-7/v-6/2

[7] Tensorflow. [Online]. Available: https://www.tensorflow.org/