A Survey of Information Acquisition in Neural Object Detection Systems

Alan Yessenbayev
A92411003
ECE Department
UC San Diego
La Jolla CA, 92092
ayessenb@ucsd.edu

Jahya Burke
A99066017
ECE Department
UC San Diego
La Jolla CA, 92092
j1burke@ucsd.edu

Nihar Wahal
A12910066
ECE Department
UC San Diego
La Jolla CA, 92092
nwahal@ucsd.edu

Sayan Mondal
A53250015
MAE Department
UC San Diego
La Jolla CA, 92092
samondal@eng.ucsd.edu

Abstract

Our goal is to assess neural network architectures in the task of object detection, followed by appropriate image segmentation. We evaluated Faster RCNN and YOLOv3 models for various metrics.

1 Introduction

Object detection is widely used in computer vision applications, usually alongside image classification, to interpret all the data a given image contains. Most applications of object detection are in automation, such as self-driving cars that can identify where different cars are in front of it, and smart factories where a robotic arm can differentiate individual parts and identify which one it needs to grab. The goal of object detection in image processing is to identify what distinct objects appear in a given image. The output of an object detection system should be the input image with bounding boxes overlaid on it which contain each distinct object. The main challenges involved with object detection pertain to the complexity of the inherent task. The network needs to first determine whether an object exists within the frame, then find the edges of the object within the frame, and then repeat this process until all the objects are found. Additionally, the effectiveness of an object detection network is heavily affected by the size of the window scanning through the image. If the window is too large, the network can miss smaller objects within the image. However, using smaller windows increases training time and has the potential to miss very large objects in the image. We approached these challenges using two different methods, YOLOv3 and Faster R-CNN networks, to explore their effectiveness.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

2 Methods

2.1 YOLO

2.1.1 Background

YOLO is a new approach to real-time object detection using the philosophy "You Only Look Once". YOLO is a single CNN that simultaneously covers object detection and object identification, as opposed to previous approaches which split the detection and identification into two separate networks. Combining both tasks into a single network significantly decreases training and testing time as each input image is only processed once instead of twice. Originally, the YOLO method for object detection was to split the input image into a preset number of smaller cells. The classifier looks at each individual grid cell, predicts where distinct objects (if any) fall within the cell, and where the bounding boxes for each object is located. Additionally, the classifier also assigns a class probability for each object it finds to predict what the object is. The first iteration of YOLO had trouble finding multiple small objects in a group (such as a flock of birds), was prone to localization errors, and had poor performance when working with input data whose objects were placed in new ways that were not present in the input data. Additionally, the original YOLO network could not identify more than 98 bounding boxes in an image due to the method used to split the input image. Overall, YOLO had less mAP than other object detection networks such as Faster R-CNN and SSD networks.

YOLOv2 improved the algorithm to increase the mAP to match these other networks. To increase the mAP, YOLOv2 introduced batch normalization to all convolutional layers within the network and increased the resolution of the classifier - collectively, these increased the mAP by roughly 6%. Additionally, YOLOv2 started using anchor boxes, similar to Faster R-CNN networks, which allowed the network to identify over 1000 bounding boxes per image. YOLOv2 also used a smaller feature extractor than YOLOv1. Collectively, these two changes significantly improved YOLO's performance in identifying multiple small objects. To improve the last major issue with YOLOv1, YOLOv2 added a randomization factor during training to the dimensions of the input data, which increased accuracy in object detection of images of varying resolution. Lastly, YOLOv1 used the Darknet-19 model as the base of its classifier, which had improved performance over the original DarkNet classifier.

In 2018, YOLOv3 was developed and featured further small improvements to the YOLO architecture. This version changed the prediction method for bounding boxes, now predicting only 1 bounding box for each object to compare against the ground truth as opposed to finding multiple potential boxes. YOLOv3 also changed from using a singular classifier with a softmax layer applied to the output to using an individual classifier for each potential class. This increased accuracy for detecting objects belonging to multiple classes. For example, while a softmax classifier might label a photo of a dog with "dog=0.4" and "poodle=0.45". In this case, an unrelated class could be chosen as the prediction because the main label was split into two classes. Individual classifiers for each object removes this issue entirely, as the network will recognize the object as "dog=True" and "poodle=True". YOLOv3 refined the scaling randomization from v2, running the object detection method on 3 specific versions of the input image. The first version just does classification on the output of the feature extractor for the input image. The second version upsamples by 2x the output from the third last feature map, combines it with a feature map from earlier in the network, and performs classification on that data. This method is repeated again on an earlier feature map in the network. The base of the classifier was changed once again to Darknet-53, which had increased performance than Darknet-19 while still being faster than ResNet-101 and ResNet-152, the previous best classifiers available. The architecture for the YOLOv3 network is shown in Figure 1 and the architecture for the Darknet-53 classifier is detailed in Figure 2 [7][2].

2.1.2 Specifications and Network Methods

One major advantage YOLOv3 has over other networks is that it only scans an image once and only predicts one bounding box per object. To achieve this, the YOLOv3 network uses logistic regression while training to determine the locations of proposed bounding boxes. If a bounding box overlaps with the ground truth box more than any other box, then the bounding box is given an "objectness" rating of 1, indicating the box very likely contains an object, and is used as the input to the classifier. Boxes which somewhat overlap with the label are given an objectness rating proportional to the classifier box but are not sent to the classifier. Boxes with an objectness value lesser than a specific threshold are discarded, and a weighted average is taken of the rest of the bounding boxes to determine the

coordinates for the final prediction. Each box has 5 identifying values: the coordinates of the corners and the center. These values are compared to the ground truth, and the loss is computed using the sum of squared error.

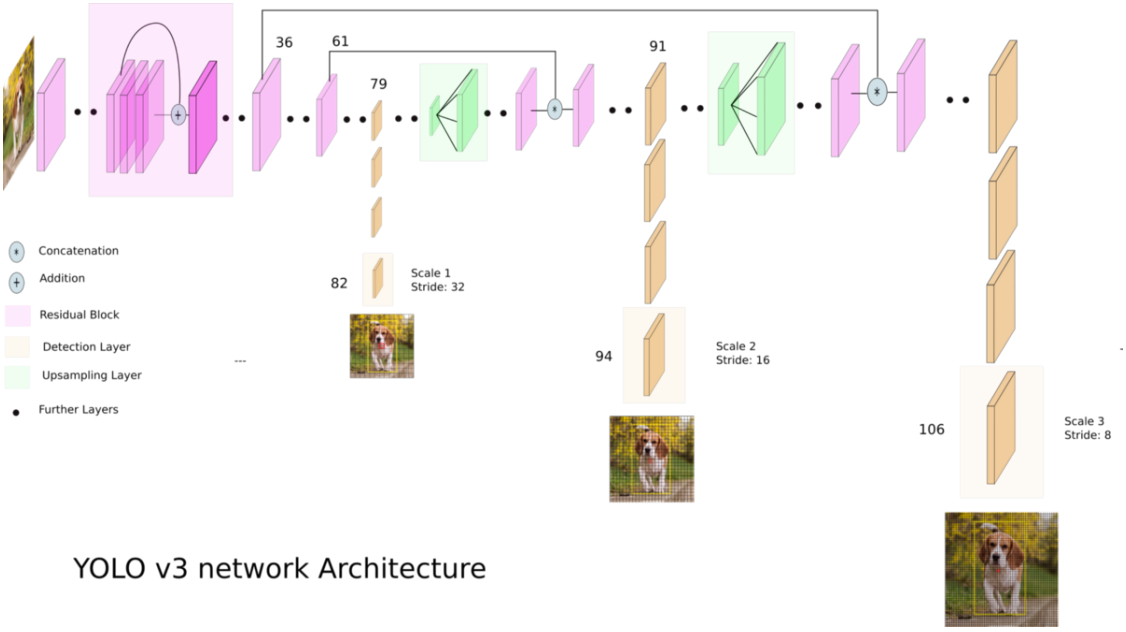

Figure 1: YOLO v3 Architecture [7]

2.2 R-CNN

2.2.1 Background

R-CNN, or Region-CNN, is another method to do object recognition with machine learning. R-CNNs split an input image into several regions using a complex algorithm called selective search. The selective search algorithm first splits the image into many very small sections. Then, using a greedy algorithm approach, similar sections are grouped together to form 2000 region proposals that could contain an object. The region proposals are reshaped to form a square and are processed by a CNN. The CNN outputs a 4096-dimensional feature vector which is then processed by an SVM classifier. The SVM algorithm takes in an input feature vector and predicts whether or not an object is present in the region. Additionally, if it does predict there is an object, then the algorithm also outputs offset values for how far the object is from the center of the region. This process is repeated for all 2000 regions, and the bounding boxes of objects are computed using the offset values. When the R-CNN was developed, it was faster to train than other object detection methods available, but it still is a very slow method as the SVM has to classify 2000 regions. Testing was slow as well, taking roughly 50 seconds to process an image. Lastly, the selective search algorithm does not change as it processes more images, so it does not adapt to the training images.

This algorithm was improved with the development of Fast R-CNN. The main difference in the new algorithm was a change in the ordering of steps. The entire image is first processed by a CNN that outputs a feature map for the input image. Then, the feature map is split into regions using the selective search algorithm. Each region is reshaped to a square, processed by a pooling layer into a fixed size square, and converted into a feature vector. This feature vector is passed through a softmax layer to classify if the region contains an object. Additionally, the feature vector is sent through another regression algorithm to determine the offsets for the regions. Since the convolution is only done once on the input image, this algorithm is considerably faster than the original R-CNN algorithm. Testing time for an input image dropped to 0.32 seconds, but the selective search algorithm still took 2 seconds.

Faster R-CNN was developed as an improvement to Fast R-CNN by modifying the region proposal method. Rather than following the selective search method, the feature map is sent to a separate

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2: Darknet-53 Architecture [2]

RPN, or Region Proposal Network, which is a deep CNN that selects regions of interest in the image. Using an RPN rather than the selective search algorithm reduced total testing time for an image to 0.2 seconds.

2.2.2 RPN Specifications

The RPN uses a sliding window to process the convolutional feature map with a small neural network according to [4]. The window is mapped to a lower-dimensional feature vector and then passed through two classifiers, one for box regression to find the corners of the bounding boxes, and the other to a box classifier to determine the box's objectness rating. The anchor points for the regions are also developed using the sliding window. Each set of anchors is a group of 9 boxes generated from the center of the sliding window. One anchor is created right at the center of the sliding window, and the other eight anchors are generated with different aspect ratios and at different coordinates relative to the first. The Intersection-over-Union, or IoU, for each anchor is computed with (1) against the ground truth boxes, and the anchors are assigned a predicted label p^* according to (2).The loss function for the RPN takes p^* and the corners of the bounding boxes t^* as inputs, and uses normalized versions of the classifier and regression loss L_{cls} (3) and L_{reg} (4) to compute overall loss. The ground truth for the label and corner coordinates is denoted as *p* and *t*.

$$IoU = \frac{\text{Area of Intersection of Region and Ground Truth Box}}{\text{Area of Union}} \tag{1}$$

$$p* = \begin{cases} 1, & \text{if } IoU > 0.7 \\ -1, & \text{if } IoU < 0.3 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Note: When computing L_{cls}, p^* and p are scaled to the range (0, 1).

$$L_{cls} = -(p^* log(p) + (1 - p^*) log(1 - p)) \tag{3}$$

$$L_{reg}(t_i - t_i^*) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{4}$$

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

N_{cls} = mini batch size

N_{reg} = number of anchor points

λ is a balancing parameter between the two loss functions (5)

3 Experimental Setup

The main metric used to compare object detection methods is mean average precision (mAP). The mAP score of a model ranges from 0 to 1 and indicates how well the model was able to correctly detect objects. The mAP score corresponds to the integral of the precision recall curve for the model. Precision recall curves show what the precision value is achieved by the model at each recall value. The formulas for precision and recall are shown in (6). To create a precision recall curve for a test set, you calculate the precision and recall for each prediction output and plot them. The mAP score is calculated by smoothing the precision recall curve and then interpolating to get average precision points for recall values of [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1].

The mAP scores and precision scores are calculated for each method. The testing set contains 300 images from the PASCAL VOC 2012 dataset with a total of 1000 bounding boxes. The pretrained model used for Faster R-CNN was trained initially on the PASCAL VOC 2007 dataset, which has the same labels as the 2012 dataset. However, the pretrained model used for YOLO was trained using the COCO dataset, which has 80 labels. The labels of the COCO dataset also include the 20 labels from PASCAL VOC. In order to run testing using this network, a dictionary was created to map the labels from COCO to PASCAL VOC. In addition, the labels for the testing set were restricted to the classes that are considered to be objects that a car may encounter while driving. Both pretrained models were evaluated to produce mAP values. The mAP standard used is 0.5, which means bounding boxes with overlaps of greater than 0.5 are considered to be a match.

In addition, an attempt was made at training each network from scratch. Due to some mathematical issues encountered, results for training YOLO were not obtained. The Faster R-CNN network was trained for 8 epochs using the PASCAL VOC Data on the subset of classes mentioned previously.The precision recall plots and mAP scores are considered for the Faster R-CNN network which was trained from scratch.

$$\begin{aligned} Precision &= \frac{TruePositives}{TruePositives + FalsePositives} \\ Recall &= \frac{TruePositives}{TruePositives + FalseNegatives} \end{aligned} \tag{6}$$

4 Results

4.1 YOLO

The resultant Precision Recall Curves for the Pre-trained YOLO Model are shown below. Due to errors with the code the precision recall plots for mAP with threshold 0.5 contained all zeros. This is

linked to the fact that we used just a subset of labels that is demonstrated in the file *labelMap.py*. To gain some insight into the issue we generated this plot for mAP score with an IoU threshold of 0. We can see that the graph is very homogeneous. The predictions had labels that matched labels in the true images but the IoU region overlap was not high enough. This could be because the YOLO images were resized and we had difficulty scaling the bounding boxes correctly to calculate IoU correctly.

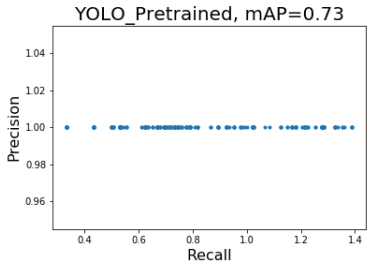

Figure 3: Precision Recall Curves for Pretrained YOLO Model

4.2 R-CNN

Using the pretrained Faster R-CNN network an overall mAP score of 0.7 is achieved. This is consistent with the general pattern of the work that we have see in the Faster R-CNN paper. When you look at the precision recall curve for the individual classes you can see a large discrepancy. However it is evident that the classes with lower mAP have far fewer predicted points in general and are under represented in the dataset. For example, the cow class has only 3 predictions.

The Faster R-CNN network that is trained from scratch has a lower mAP score of 0.5. This is due to the fact that training was only for 8 epochs. The training epochs each took roughly 1 hr to run using 70k training images from the PASCAL VOC 2012 dataset. Looking at the training loss curves for each network it is clear that the model is converging. The training loss reduces steadily while the validation loss is reducing and flattening out as is expected.

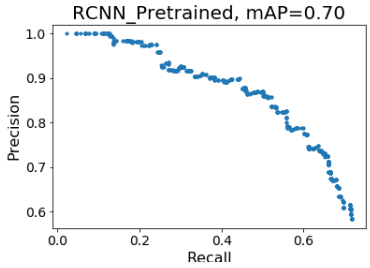

Figure 4: Precision Recall Curves for Pretrained R-CNN Model

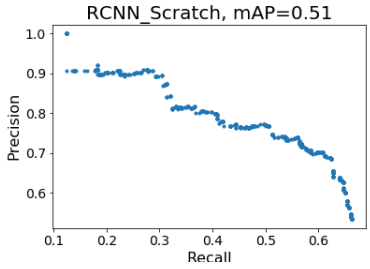

Figure 5: Precision Recall Curves for R-CNN Model trained for 8 Epochs

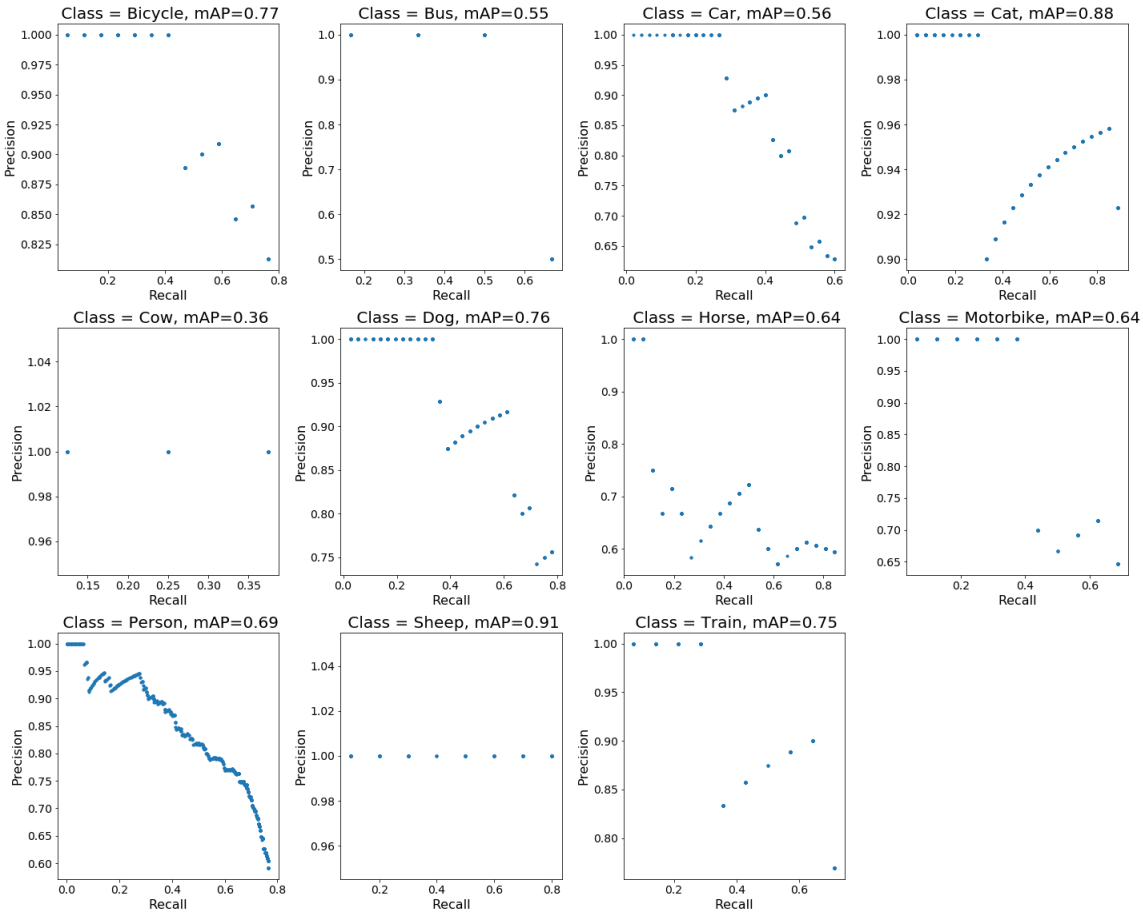

Figure 6: Class Precision Recall Curves for Pretrained R-CNN Model

5 Discussion

The precision-recall curves in Figures 4 and 5 show the training differential in R-CNNs. The curve in Figure 5 is lower than the curve in Figure 4, showing that the model when fully trained was more accurate than when it was only trained for 8 epochs. This difference is further demonstrated in the precision-recall curves for individual classes in Figures 6 and 7. The mAP for almost every class is higher for the fully trained network than the mAP for the scratch R-CNN. Additionally, the training loss plots in Figure 8 show how significantly the R-CNN model is improved with training. The first two plots show the training loss for the RPN, the next two show the same for the classifier, and the last plot shows the training loss for the R-CNN as a whole. The sharp decrease in the plots shows the loss drop as the network starts being trained.

For YOLO, we tried to get the precision-curves as well but in the process we encountered a couple of issues. First, the YOLO network was pretrained on the COCO datasets, and we tried to test it on the PASCAL VOC dataset, which had different label values for the same classes. In order to fix that we created a mapping for COCO to PASCAL VOC dataset. Next we found that the network was predicting the bounding boxes for the output feature map, which resulted in the coordinates of the bounding boxes being placed out of bounds with respect to the input image size for a few images when traced back. We fixed that problem by creating a threshold for the predicted bounding boxes to lie within the input image size. In the end, we discovered that separating tasks of detection and bounding box regression can be very fruitful, however, if we modularize the algorithm fully, we are able to achieve much faster testing performance, and this consideration is very important, application and scenario specific. The plot shown in Figure 3 shows the precision-recall curve for the pretrained

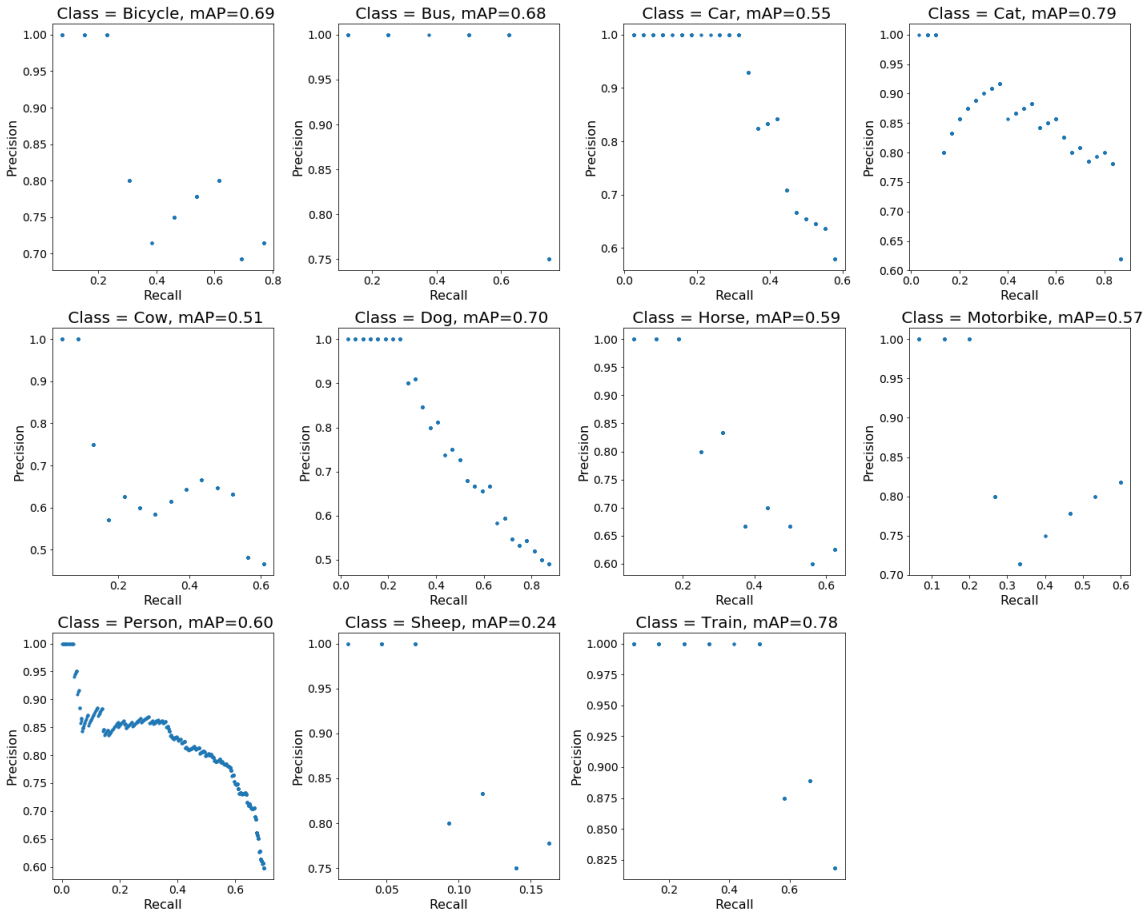

Figure 7: Class Precision Recall Curves for R-CNN Model trained for 8 Epochs

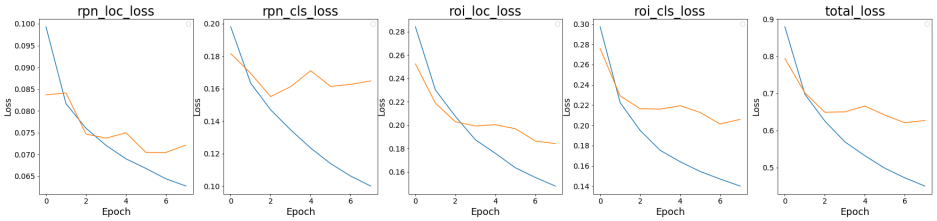

Figure 8: Training Loss Plots for R-CNN Trained from Scratch
Blue - Training Loss, Yellow - Validation Loss

YOLO network. The precision values are consistently at 1.00 across the recall values, showing that the network was almost completely accurate in its predictions.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi You Only Look Once: Unified, Real-Time Object Detection *arXiv preprint arXiv:1506.02640*, 2016

[2] J. Redmon, and A. Farhadi YOLOv3: An Incremental Improvement *arXiv preprint arXiv:1804.02767*, 2018

[3] R. Girshick, Microsoft Research Fast R-CNN *arXiv preprint arXiv:1504.08083*, 2015

[4] S. Ren, K. He, R. Girshick, and J. Sun Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks *arXiv preprint arXiv:1506.01497*, 2016

[5] A. Kathuria Series: YOLO object detector in PyTorch https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/

[6] Machine-Vision Research Group Guide to build Faster R-CNN in PyTorch, https://medium.com/@fractaldle/guide-to-build-faster-R-CNN-in-pytorch-95b10c273439?fbclid=IwAR1TFy48DnJWW9mfj-ZFo6Rjk1HTB8rXqFr-CBdEVTkbW2cFYFRKKY6lFZU

[7] A. Kathuria What's New in YOLOv3? https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

[8] R. Gandhi R-CNN, Fast R-CNN, Faster R-CNN, YOLO—Object Detection Algorithms https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[9] R. Girshick, J. Donahue, T. Darrell, J. Malik Rich feature hierarchies for accurate object detection and semantic segmentation *axXiv preprint arXiv:1506.01497v3*, 2016

[10] mc.ai YOLO3: A Huge Improvement https://mc.ai/yolo3-a-huge-improvement/