DEEP UNDERSTANDING OF YOLOV1 AND OBJECT DETECTION SYSTEM BASED ON YOLOV1

Binfang Ye
New York University

Simon Wang
New York University

1 INTRODUCTION

Deep learning in the computer vision field is developing at a rapid pace. Many car industries are combining computer vision algorithms to make intelligent vehicles such as driverless cars. However, driverless cars are not ubiquitous because it is hard to detect objects around the car correctly. Therefore, Autonomous vehicles are hard to make real-time and right decisions. To decrease the car accident rate and to protect human beings, a good object detection algorithm that can convey accurate information to humans and computers is necessary. The algorithms play an essential part in helping driverless cars "see" the objects. There are many object detection algorithms such as Faster-RCNN, SSD, etc. All object detection algorithms require a large dataset to train the model for gaining good prediction accuracy. In this course, we try to have a deep understanding of one object detection algorithms among them.

YOLO (You Only Look Once) family is one of the object detection algorithms and is proposed by Redmon et al. (2016). Unlike R-CNN, YOLO is a one-stage algorithm that is the main reason why it runs faster than other algorithms. In addition, it is important to know that the YOLO is treating object detection as a regression problem which is from image pixels to bounding box and class probabilities. In this work, we study and present a recreation of the YOLO (v1) algorithm and explain it in detail. We will provide a detailed summary of our project implementations, algorithm theory, and the result analysis. All of our source codes and files can be found in Github: `https://github.com/yeb2Binfang/ECE-GY9123_DL/tree/main/Project/Yolov1`

2 LITERATURE SURVEY

You Only Look Once(YOLO) is proposed by Redmon et al. (2016). YOLO is an object detection algorithm that aims to solve object detection problems as a regression problem. The *YOLOv1* network divides an image into many grid cells and uses local grid cell features to make predictions of bounding boxes. It predicts all bounding boxes for all classes in an image simultaneously.

Redmon & Farhadi (2017) improved the YOLO algorithm that can detect over 9000 different object categories in real time. The improved model (*YOLO9000*) can well balance running speed and accuracy. Compared with faster-RCNN and SSD, *YOLO9000* is much faster while achieving high accuracy on VOC 2007 at 67 FPS. Redmon & Farhadi (2017) proposed joint training that allows the model to predict data that do not have any annotation information and they tested the model on ImageNet validation set getting 19.7 mAP.

Redmon & Farhadi (2018) updated a new YOLO network (*darknet*) that is faster and more accurate than previous work. However, the new network is larger than the prior network. Compared with SSD, YOLOv3 can reach the same accuracy but three times faster on the same testing dataset. Compared with YOLOv1, YOLOv3 can well perform on detecting the small objects.

Bochkovskiy et al. (2020) combined different techniques including Weighted-Residual-Connections (WRC), Cross-Stage-Partial-connections (CSP), Cross mini-Batch Normalization (CmBN), Self-adversarial-training (SAT), and Mish-activation to achieve high detect accuracy. Bochkovskiy et al. (2020) developed an efficient and fast model that is able to use in the video. The model is much easier to train compared with the prior work.

3 YOLOv1 ALGORITHM

Yolov1 is an end to end algorithm which puts the resized image to the Yolov1 network and output the thresholded bounding boxes image. Figure 1 shows how does the YOLOv1 process the images.

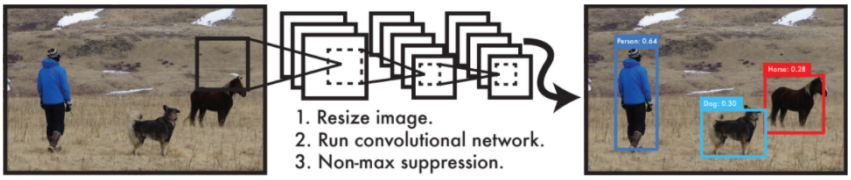


Figure 1: The overview of YOLOv1 (Redmon et al., 2016)

3.1 NETWORK EXPLANATION

Yolov1 Network is large, deep, and complicated. Figure 2 shows the YOLOv1 Network Architecture. In our implementation, we constructed the deep neural network with 24 convolutional layers, included 4 max-pool layers, and followed 2 fully connected layers at the end.

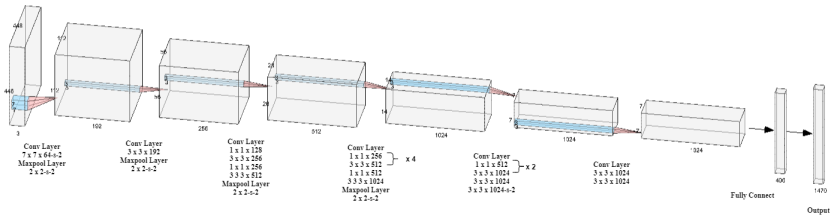


Figure 2: The YOLOv1 network architecture

The original paper used 4096 neurons for the fully connected layer1. However, to save training time and GPU space, we changed it to 496 neurons as shown in Figure 2. Max pooling layer is able to reduce the training dimensions and we can notice that in the network, 1×1 convolution is used often since it can increase or decrease channels easily without using many parameters. Therefore, using 1×1 convolution in YOLOv1 network is able to reduce the network complexity. We utilized zero-padding to ensure a more consistent layer output shape. Inputs to this network are expected to be RGB images of shape $(448 \times 448 \times 3)$. The details of the network architecture can be found in table 1. It's worth mentioning that layer clusters, like Conv7 to Conv14, consist essentially of repetitions of two consecutive layers (in this case, Conv7 and Con8). The purpose of this structure is to expand the number of trainable parameters to improve the network's overall accuracy. Besides, it can help the network capture more features from the training data due to the training set containing massive information. Output of this network is a tensor of shape $7 \times 7 \times 30$. So, for the output layer shown in Figure 2, we need to reshape it to the shape we expected. The original $448 \times 448 \times 3$ input image is divided into 7×7 grid cells, each of which contains a 30-element array. Indices 0-19 represent 20 class conditional probabilities $P(Class_i|Object)$. Indices 20-24 and 25-29 each indicate a prediction bounding box's information:

- x: x-coordinate of object detector center, normalized with respect to the width of grid cell.
- y: y-coordinate of object detector center, normalized with respect to the height of grid cell.
- w: width of object detector, normalized with respect to the image width.
- h: height of object detector, normalized with respect to the image height.
- c: confidence of object detector.

Finally, class conditional probabilities from Output[:, :, 0:19] and confidences from Output [:, :, 24] and Output [:, :, 29] will determine the probability of the predicted class existing in the grid cell and how well the predicted x, y, w and h match the object.

3.2 LOSS FUNCTION OF YOLOV1

The final layer predictions contain class probabilities and bounding box information. We will pick the maximum probability as the final prediction. We also normalize the weight w, height$h$, x, and y falling between 0 and 1. As mentioned, Yolov1 reframes object detection as the regression problem. Therefore, it is easy to use sum-squared error to optimize our model. The loss function used in our model is

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2] \tag{1}$$

$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w}_i - \sqrt{\hat{w_i}^2}) + (\sqrt{h_i} - \sqrt{\hat{h_i}})^2] \tag{2}$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C_i})^2 \tag{3}$$

$$+\lambda_{noobj} \sum_{i=0}^{S^B} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C_i}) \tag{4}$$

$$+ \sum_{i=0}^{S^2} 1_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \tag{5}$$

Part(1) and part(2) is penalizing the coordinate error and the ratio error. For part(2), the sum-squared error will equally weight errors in large boxes and small boxes if we do not add square root on w and h which will result to badly align with the maximizing average precision. The small boxes should matter more than the large boxes when we use the same deviations on them. Taking the square root can partially address this issue. Figure 3 shows how does the square root work. Part(3) and part(4) will penalize the confident errors. But for part(3), we take the bounding box that is responsible for the predicting object, while for part(4), we take the bounding box that is not responsible for the predicting object. In the loss function, 1_{ij}^{obj} denotes that the j^{th} bounding box predictor in cell i takes the responsible for the prediction. We will only take only one bounding box that has the highest IOU with the ground truth in each cell to be responsible for predicting object. We use two parameters $\lambda_{cood} = 5$ and $\lambda_{noobj} = 0.5$ to make the model stable. Many grid cells in a image probably do not have any objects so that the confident scores for those cells are 0, which will affect the gradient from cells that do have the object. Therefore, we increase the loss for cells that do contain the object and decrease the confident loss from cells that do not contain objects. Part(5) will penalize the classification error. 1_{i}^{obj} denotes that whether an object appears in the cell it or not.

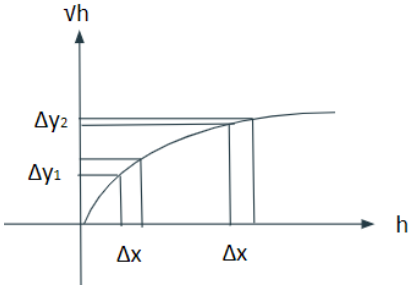


Figure 3: It shows that the small boxes matter more than the large boxes if we add square root in the loss function.

3.3 IOU AND NON-MAXIMUM SUPPRESSION

YOLOv1 uses interception-over-union (IOU) to calculate the accuracy of the object detector. Figure 4 shows the way to calculate the IOU value, using coordinates of a pair of diagonal corners of the ground true box and predict box. We need to set an IOU threshold value to measure if the result of the predicting box is correct or not. In addition, the algorithm will generate many bounding boxes at the end. However, not all of the bounding boxes are necessary to show up because many of them point to the same object or no-object areas. Therefore, we need to suppress unnecessary bounding boxes. We will set a threshold σ to suppress the useless boxes that the predicting score is lower than the threshold value.

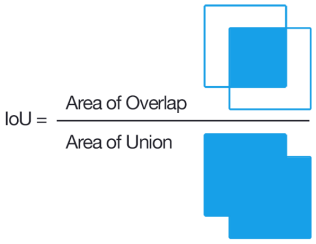


Figure 4: It shows how does interception-over-union work.

4 IMPLEMENTATION DETAILS

4.1 DATASET

Our dataset of choice is the PascalVOC dataset that contains 43,223 images. This dataset offers a huge number of annotated images and we will focus on using its "Bounding Box" annotations. An annotation TXT file is also provided with useful information including "class", "x", "y", "w", "h". During our training process, we will use such information to identify if the object(s) within an image and their locations in terms of XY coordinates. We can also identify the bounding boxes through ratio W and H. There are 20 object categories in total. The number from [0,19] represents airplane, bike, bird, boat, bottle, bus, car, cat, chair, cow, table, dog, horse, motorbike, person, plant, sheep, sofa, train, and tv differently. However, we found that the data and labels (9,958 images and labels are matched correctly) are messy in this dataset. Therefore, we cleaned up the dataset and randomly selected 5,000, 200, and 100 images as the training, validation, and testing dataset respectively. The dataset, label information, and the CSV files can be found `https://drive.google.com/drive/folders/1dqW6nx5gaRX-XihvkK81Kw1xhsT0j9Sr?usp=sharing`

4.2 TRAINING

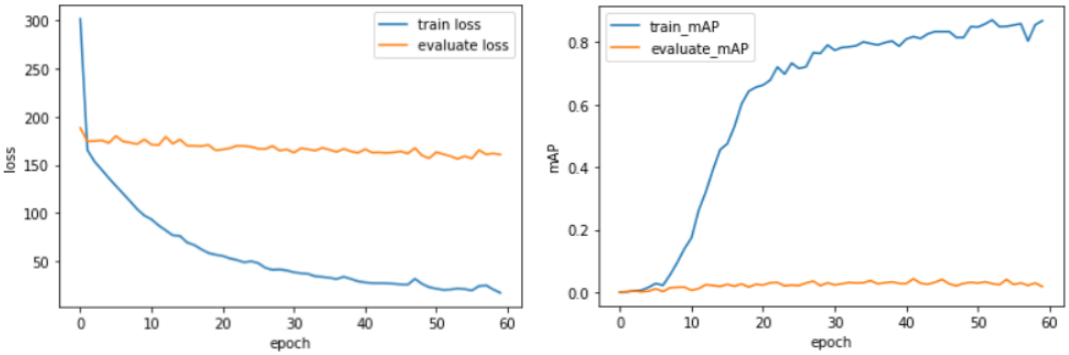
The final layer uses liner activaiton function and for other layers, leaky ReLU is used. The leaky ReLU is defined as:

$$\phi(x) = \begin{cases} x & \text{if x is greater than 0} \\ 0.1x & \text{otherwise} \end{cases}$$

We iteratively train the YOLO model defined in section 3 with the PascalVOC dataset. We use an Adam optimizer with a 2e-5 learning rate and zero-weight decay. As model forward and loss function have been introduced, we proceed to a simple backpropagation to train our model. Due to our limited computational power, we choose a training batch size of 16, instead of 64 as done in Redmon et al. (2016)'s work. We set the IOU threshold and the non-maximum suppress threshold value to 0.5 and $\sigma = 0.4$ respectively. We performed the YOLOv1 network on Google colab with Tesla T4 GPU. The model was trained with 5000 images and corresponding label information recorded in the CSV and txt files. The model was trained for 60 epochs and once its training Mean Average Precision reaches a threshold value (0.85), current model parameters will be automatically saved into local storage. The trained model (5000 images) can be found: `https://drive.google.com/drive/folders/1gYuX5FztKkzlj4W6k8ohfMfsuff0MbAv?usp=sharing`

5 RESULTS

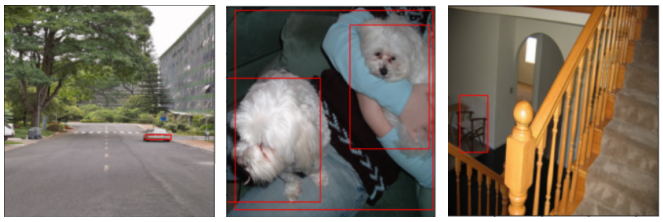
We utilize the YOLO loss function defined in section 3.2 and Mean Average Precision (mAP) as our evaluation metrics. The Mean Average Precision is a measure of the accuracy of both categorical and locational accuracy of image classification. The mAP score is calculated by averaging AP over all classes. While YOLO loss primarily focuses on perfecting the CNN, the mAP score serves as a validation tool to visualize how accurate the model's current prediction is, compared to the image labels. From Figure 5, we can notice that the training loss is decreasing while the evaluation loss decreases a little bit. The mAP of training is increasing while the mAP of evaluation increases a little bit. The reason why it happens is that the training dataset is far from enough so that the CNN cannot capture sufficient information to train the model and because of that, it results in a bad performance model. Therefore, we have better collect more usable annotated images to train the network. Redmon et al. (2016) pretrained the model on the ImageNet dataset and they used more than a week to train the network.



(a) Loss (b) mAP

Figure 5: a) is the Loss plot and b) is the mAP plot

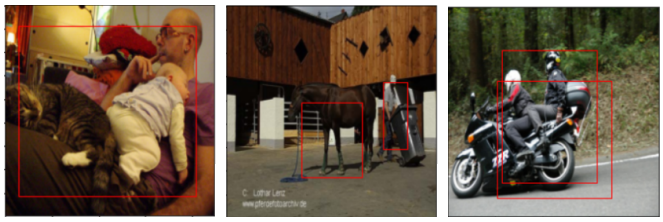
We apply the model to the training, evaluation, and testing dataset to check how does the model perform. It is reasonable that the model can well perform on the training dataset even though there is a very small object since the model is overfitting on the training dataset. In the notebook, it will print out the corresponding predicting information in [classification, confidence score, x, y, w, h] format. For example, Figure 6 (a) shows the data [6.0, 1.0500506162643433, 0.7331274151802063, 0.6220758557319641, 0.12114816904067993, 0.023689648136496544]. And number 6 denotes car as mentioned in section **4.1**



(a) car(small) (b) dogs and person (c) chair

Figure 6: Correct predictions on training dataset

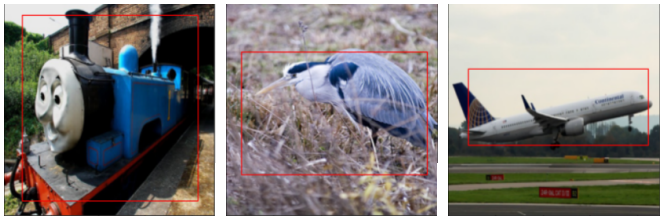
We evaluate the model on 200 images (evaluation dataset) during training. However, Figure 5 shows the evaluation mAP is quite low. Therefore, it is interesting to visualize the model performance on the evaluation dataset. Figure 7 shows correction detections on evaluation dataset. We can notice that the model can well detect the object that is large in the image. The detector will get a low confidence score on the object if is a medium size in the image. For example, for the confidence score, Figure 7 (b) got 0.4859876036643982 and (a) got 0.625907301902771.



(a) person (b) horse (c) motorbike

Figure 7: Correct predictions on evaluation dataset

We try to use big object images to confirm the conclusion we got from the model performance on the evaluaiton dataset. Therefore, we randomly picked some big objects from the testing dataset. We are interested to know how well does the trained model perform on big object images. Figure 8 shows the correct detection on testing dataset. We can notice that the model can well detect as long as the object is large even though the model is not well-trained enough.



(a) train (b) bird (c) airplane

Figure 8: Correct predictions on testing dataset

However, there are many wrong detections. We found three main errors that the detector will make often. The errors are wrong predictions, no-boxes at all, and repeat bounding boxes. Figure 9 shows the three cases. For (a), the model denotes the object as 6 (car) but the true value is 13 (motorbike), even though the object is large. In image (a), we found that this motorbike is similar to a car. The easy-trained model cannot differentiate similar objects. For (b), there are no boxes at all. We notice that every object in image (b) is kind of small. The model is hard to detect the small object. For (c), we can see that many repeated bounding boxes frame the same object. The reason is that the closed responsible bounding boxes have almost the same confidence score. There three denoted person bounding boxes in (c) and the confidence scores are 0.5776211023330688, 0.4733344614505768, 0.4712468683719635. Therefore, it is hard to suppress the repeated bounding box but one way is to adjust the non-maximum suppress threshold value.

(a) wrong (b) no boxes (c) repetition

Figure 9: Wrong detections on randomly picked images

6 Conclusion and Future Directions

In this work, we successfully created an affordable version of the YOLOv1 network. Our model showed promising prediction results for identifying different classes of objects within images. The

Layer Name	Filter Shape	Zero Padding	Stride	Output Shape
Conv1	7 x 7 x 64	3	2	224 x 224 x 64
MP1	2 x 2	0	2	112 x 112 x 64
Conv2	3 x 3 x 192	1	1	112 x 112 x 192
MP2	2 x 2	0	2	56 x 56 x 192
Conv3	1 x 1 x 128	0	1	56 x 56 x 128
Conv4	3 x 3 x 256	1	1	56 x 56 x 256
Conv5	1 x 1 x 256	0	1	56 x 56 x 256
Conv6	1 x 1 x 512	0	1	56 x 56 x 512
MP3	2 x 2	0	2	28 x 28 x 512
Conv7	1 x 1 x 256	0	1	28 x 28 x 256
Conv8	3 x 3 x 512	1	1	28 x 28 x 512
Conv9	1 x 1 x 256	0	1	28 x 28 x 256
Conv10	3 x 3 x 512	1	1	28 x 28 x 512
Conv11	1 x 1 x 256	0	1	28 x 28 x 256
Conv12	3 x 3 x 512	1	1	28 x 28 x 512
Conv13	1 x 1 x 256	0	1	28 x 28 x 256
Conv14	3 x 3 x 512	1	1	28 x 28 x 512
Conv15	1 x 1 x 512	0	1	28 x 28 x 512
Conv16	3 x 3 x 1024	1	1	28 x 28 x 1024
MP4	2 x 2	0	2	14 x 14 x 1024
Conv17	1 x 1 x 512	0	1	14 x 14 x 512
Conv18	3 x 3 x 1024	1	1	14 x 14 x 1024
Conv17	1 x 1 x 512	0	1	14 x 14 x 512
Conv20	3 x 3 x 1024	1	1	14 x 14 x 1024
Conv21	1 x 1 x 512	0	1	14 x 14 x 512
Conv22	3 x 3 x 1024	1	2	7 x 7 x 1024
Conv23	3 x 3 x 1024	1	1	14 x 14 x 1024
Conv24	3 x 3 x 1024	1	1	14 x 14 x 1024
FC1	/	/	/	496
FC2	/	/	/	7 x 7 x 30

Table 1: The network contains 24 convolutional layers followed with 2 fully-connected layers. The table clearly describes the network components and shows how does the network come to 7 x 7 x 30 as the final result.

model can work better if the object is large in the image. In this version of the implementation, the output of the Neural Network is in shape 7 x 7 x 30 and, in these 30 channels, only 20 are responsible for encoding the category of the object of interest, therefore a YOLO network can only detect 20 classes of object. At the same time, we designed each pixel to be contained in a maximum of 2 bounding boxes; this indicates that if three objects exist in the same frame of the image and occlude one another in front of the camera view, YOLO is likely to perform object detection in an unsatisfactory manner. Our easy-trained model has some obvious limitations. It is hard to detect similar objects such as motorbike and car. In addition, a small object is difficult to be detected. Lastly, the model does not remove the repeated bounding box. We propose some methods to address the limitations. Firstly, increasing the training dataset can improve model performance (high evaluation mAP) since the CNN can capture more information. Secondly, increasing the grid cells instead of using 7×7 is worth trying to detect the small objects. Lastly, using a suitable non-maximum threshold to remove unnecessary bounding boxes.

In addition, we provided enough details to explain the YOLOv1 algorithm in this work. It will help us understand the latter version of YOLO algorithms. For future work, we will try the techniques we presented above and boost the model performance.

REFERENCES

Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.