Deep Understanding of YOLOv1 and Object Detection System Based on YOLOv1

Binfang Ye(@by2034)
Simon Wang(@ssw8641)

1 Problem Statement

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. YOLO (You Only Look Once) is one of the object detection algorithms and plays an essential part in helping driverless cars "see" the objects. In this work, we present a recreation of the YOLO (v1) algorithm.

2 Literature Survey

In 2015, Redmon, et al. introduced *You Only Look Once*(YOLO) object detection algorithm that aims to solve object detection problems as a regression problem. A YOLO (v1) 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.

3 Dataset Description

Our dataset of choice is Pascal_VOC 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 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.

4 YOLOv1 Network

Yolov1 Network is large and complicative. Figure 1 shows the YOLOv1 Network Architecture

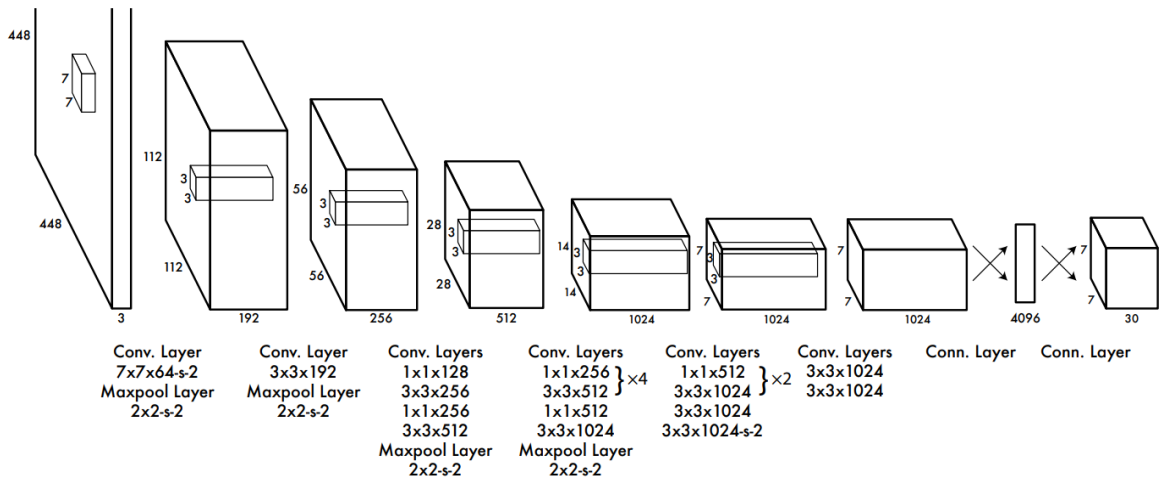

Figure 1. YOLOv1 net

We recreate the Yolov1 Network. Figure 2 shows the details of the network.

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
Conv19	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 1024
Conv22	3 x 3 x 1024	1	2	7 x 7 x 1024
Conv23	3 x 3 x 1024	1	1	7 x 7 x 1024
Conv24	3 x 3 x 1024	1	1	7 x 7 x 1027
FC1	/	/	/	4096,
FC2	/	/	/	7 x 7 x 30.

Figure 2. Yolov1 network details

4.1 Network Explanation

In our implementation, we constructed our deep neural network with 24 convolutional layers, 4 max-pool layers and 2 fully connected layers. We utilized zero-padding to ensure a more consistent layer output shape. Inputs to this network are expected to be RGB images of shape ($448 * 448 * 3$). The network's detailed architecture can be found in Figure 2. 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.
Output of this network is a tensor of shape 7730. The original 448448 input image is divided into 77 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 center, normalized with respect to the width of grid cell,

y: y-coordinate of object center, normalized with respect to the height of grid cell,

w: width of object, normalized with respect to the image width,

h: height of object, normalized with respect to the image height,

c: confidence of object.

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 fit the object.

4.2 Loss function

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

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

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

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

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

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

Part(I) and part(II) is penalizing the coordinate error and the ratio error. For part(II), 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. 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 2 shows how does the square root work. Part(III) and part(IV) will penalize the confident errors. But for part(III), we take the bounding box that is responsible for the predicting object, while for part (IV), 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 many 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(V) will penalize the classification error. 1_{i}^{obj} denotes that whether an object appears in the cell it or not.

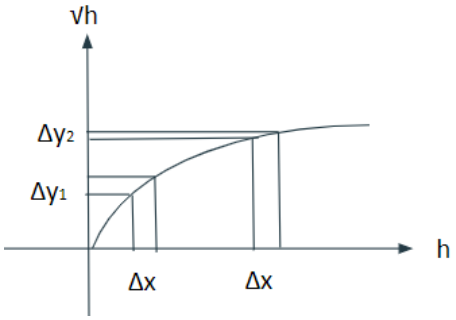

Figure 3: It shows that the small boxes matter more than the large boxes if we add square root in the loss function.

5 Preliminary Results

As of April 22nd, we have successfully implemented model architecture, training procedure and loss function of our network. These files can be found under the Github repository: https://github.com/yeb2Binfang/ECE-GY9123_DL/tree/main/Project/Yolov1, in files model.py, train.py and loss.py, respectively.

6 Remaining Work and Possible Challenges

The next step we will do is that we will train our network on NYU HPC and apply the trained model to the real world pictures. We will compare with the other object detection algorithms if it is possible. Our current possible challenge is primarily our computational power. It is also difficult for us to estimate what sort of GPU we need and its resulting training time in our scenario since we need to train a convolutional neural network with over 20 layers with a 4G training data.

7 References

https://www.kaggle.com/dataset/734b7bcb7ef13a045cbdd007a3c19874c2586ed0b02b4afc86126e89d00af8d2

PASCAL_VOC Dataset

https://arxiv.org/pdf/1506.02640.pdf

You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon, Santosh Divvala, Ross Girshick , Ali Farhadi

https://github.com/abeardear/pytorch-YOLO-v1

Guide on YOLO V1 implementation