Semantic Grouping Classification Labels for Yolo Object Detection Framework.

By

Youssef Sbeiti

Karim Hasbini

Group 20

Abstract

The purpose of this project was to look into the “You Only Look Once” (YOLO) [1] object detection framework. More specifically, we tried to find a correlation between reducing the number of labels by semantical grouping would affect performance measured with the Mean Average Precision metric (mAP). The validity of our results was bounded by the performance of our implementation of the YOLO framework. We recognize that re-implementing the framework ourselves, as opposed to using the authors’ official repository was a mistake and we believe our results are not reflective of the Yolo framework as an idea, but rather our implementation of it.

Introduction

Object detection is a well studied topic in computer vision and machine learning. The purpose of object detection is to localize and classify objects given a single image as input. Localizing the object is done by regressing bounding box coordinates. There are multiple ways to encode the coordinates of a bounding box. In our case, we followed the convention from the authors of YOLO which is: x, y, w, h . Where $x \text{ and } y$ represent the centers of the bounding box in relative image dimensions(i.e. scaled to 0-1 range). And $w \text{ and } h$ are the width and height of the box respectively, also in relative image dimensions. Classification is done by predicting class probabilities across all labels, and an “Objectness” score for each . In total, each bounding box would have 5+C corresponding values : x, y, w, h, objectness_score and a One Hot label encoding vector of length C with C being the number of classes. The main goal of this project was to see whether reducing the number of classes for the same dataset would yield better detection results. Our reasoning being that reducing the number of classes would make the surrogate loss function easier to optimize and there resulting in better predictions

Methodology and Implementation

For the purposes of the project, we decided to implement the Yolo framework in Tensorflow.js using the Layers API which made it easy to recreate the tiny Yolo v2 Convolutional Neural Network architecture (figure 1). The authors of the original paper made weights for a pre-trained model publicly available on their website https://pjreddie.com/darknet/yolov2/ . We used these weights to initialize our Network’s convolution kernels. Batch normalization parameters were initialized randomly.

Another part of implementing the Yolo framework, was the creation of the target labels that the Network would output. We used a public Chess dataset provided for free by Roboflow. We trained our model on two versions of the dataset. Both versions contained the same images but they were annotated differently. One dataset, that will be referred to as dataset A, contained 12 class lables, while dataset B contained only two classes as seen in table-1.

Dataset A classes	Dataset B classes
Black Bishop	Black Piece
Black King	
Black Knight	
Black Pawn	
Black Queen	
Black Rook	
White Bishop	White Piece
White King	
White Knight	
White Pawn	
White Queen	
White Rook	

Table-1. Class mapping between datasets

The dataset provided us with text file annotations that we had to parse and create the final target tensors. The model outputs labels as tensors of the following shape:

$$S \times S \times A \times (5 + C)$$

Where S is the is the number of grid cells per row/column as shown in figure-2, and C is the number of different classification labels. A is the number of predefined box priors that are used within every cell to make it possible to predict objects with different aspect ratios. Anchor boxes were introduced in the second iteration of Yolo [2]

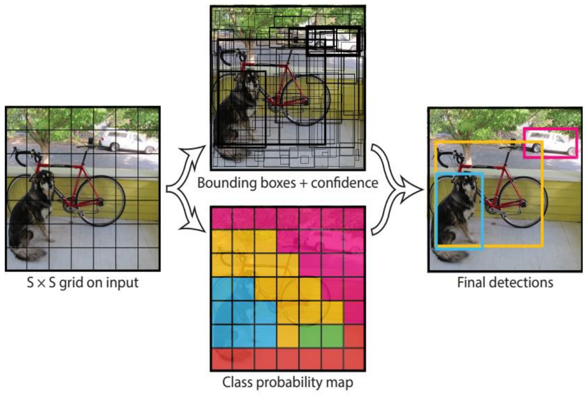
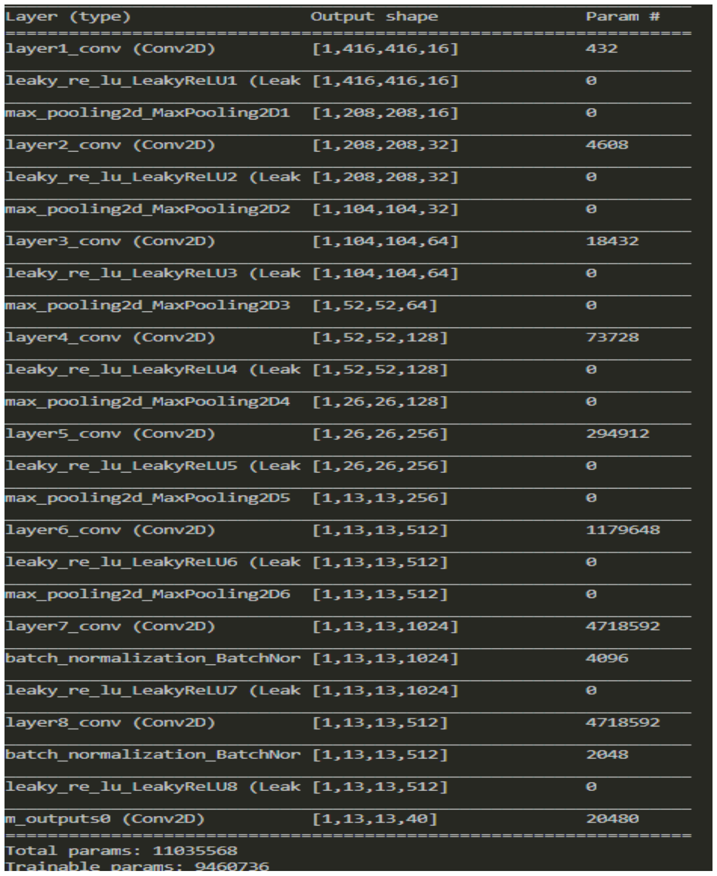


Figure 2-Yolo image grid [1].



```
Layer (type)                 Output shape              Param #
=================================================================
layer1_conv (Conv2D)         [1,416,416,16]            432
leaky_re_lu_LeakyReLU1 (Leak [1,416,416,16]            0
max_pooling2d_MaxPooling2D1  [1,208,208,16]            0
layer2_conv (Conv2D)         [1,208,208,32]            4608
leaky_re_lu_LeakyReLU2 (Leak [1,208,208,32]            0
max_pooling2d_MaxPooling2D2  [1,104,104,32]            0
layer3_conv (Conv2D)         [1,104,104,64]            18432
leaky_re_lu_LeakyReLU3 (Leak [1,104,104,64]            0
max_pooling2d_MaxPooling2D3  [1,52,52,64]              0
layer4_conv (Conv2D)         [1,52,52,128]             73728
leaky_re_lu_LeakyReLU4 (Leak [1,52,52,128]             0
max_pooling2d_MaxPooling2D4  [1,26,26,128]             0
layer5_conv (Conv2D)         [1,26,26,256]             294912
leaky_re_lu_LeakyReLU5 (Leak [1,26,26,256]             0
max_pooling2d_MaxPooling2D5  [1,13,13,256]             0
layer6_conv (Conv2D)         [1,13,13,512]             1179648
leaky_re_lu_LeakyReLU6 (Leak [1,13,13,512]             0
max_pooling2d_MaxPooling2D6  [1,13,13,512]             0
layer7_conv (Conv2D)         [1,13,13,1024]            4718592
batch_normalization_BatchNor [1,13,13,1024]            4096
leaky_re_lu_LeakyReLU7 (Leak [1,13,13,1024]            0
layer8_conv (Conv2D)         [1,13,13,512]             4718592
batch_normalization_BatchNor [1,13,13,512]             2048
leaky_re_lu_LeakyReLU8 (Leak [1,13,13,512]             0
m_outputs0 (Conv2D)          [1,13,13,40]              20480
=================================================================
Total params: 11035568
Trainable params: 9460736
```

Figure 1-Network architecture

To train the model, the following cost function taken from the original paper[1] was implemented.

$$\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \qquad (3)
\end{aligned}$$

The first two terms in the loss function penalize wrong bounding box predictions. Note that $x_i, , y_i$ as shown in the cost function are coordinates of the boxes center relative to the grid cell it is “assign to”. These coordinates are shown in figure-3 as t_x, t_y. Additionally, the model does not directly predict the boxes width and heights. It predicts scaling factors of the Anchor box that the target box is "assigned to". This can be seen in more detail in figure-3 as well, where $t_w and\ t_h$are the numbers predicted directly by the model and represented by w_i, h_i in the cost function above. The last three terms penalize wrong classification and high confidence scores in the wrong grid cells. $\lambda_{coord}\ and\ \lambda_{noobj}$ are factors to tune the directly related to increasing bounding box prediction accuracy, and supressing erroneous high score predictions respectively. More details about the loss function can be found in the authors' paper[1][2].

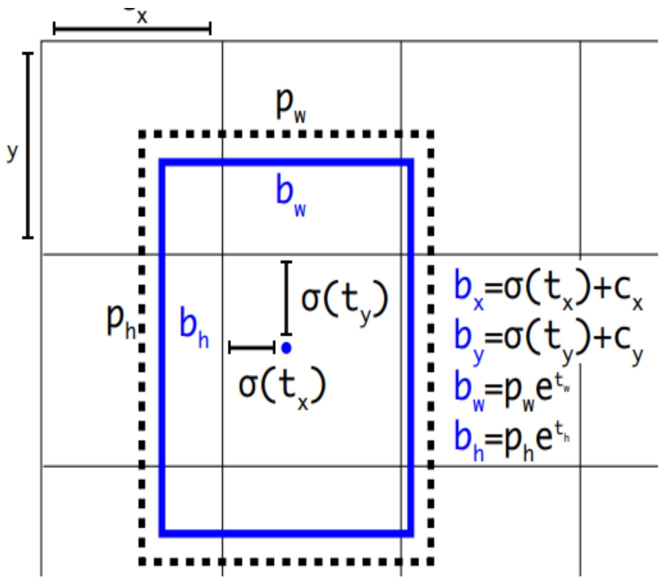


Figure 3-Yolo Anchor boxes [2

<u>Model Evaluation and Results</u>

Usually, for Object Detection algorithm, the final metric used for evaluating performance is the Mean Average Precision (mAP) over all possible classes. Average Precision is the Area Under the Curve of the Precision x Recall curve[3]. The mean of the Average precision for each class yields the mAP. mAP is usually accompanied by an IOU threshold. For example, it can be written as mAP@50, meaning mAP at an IOU threshold of 50%. This is a threshold to distinguish between True predictions and False ones. In other words, if a predicted box has an IOU of less then the IOU threshold with the ground truth box it is classified as a False Positive. Otherwise, if the IOU between the predicted and the ground truth box is more then the threshold, it is then considered a True Positive. More information about the mAP metric in this survey[3].

	A	B
mAP@25	59.67%	40.66%
mAP@50	48.58%	32.64%
mAP@75	5.18%	2.94%

Table-2: Results

From Table-2, we can see that our experiment results do not align with our hypothesis. The same Neural Network got better results on dataset A that contains 12 labels. Upon further inspection of the results in figures 4 and 5, we can see that our model did a good job classifying objects of Black color with an Average Precision of 48.34%, while for White pieces it only got 16.94%.

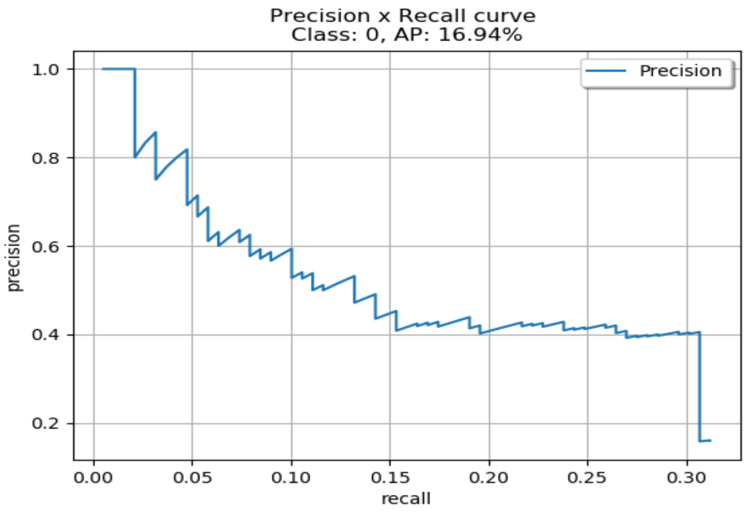


Figure-4: Precision x Recall Curve for class White-Piece on test set

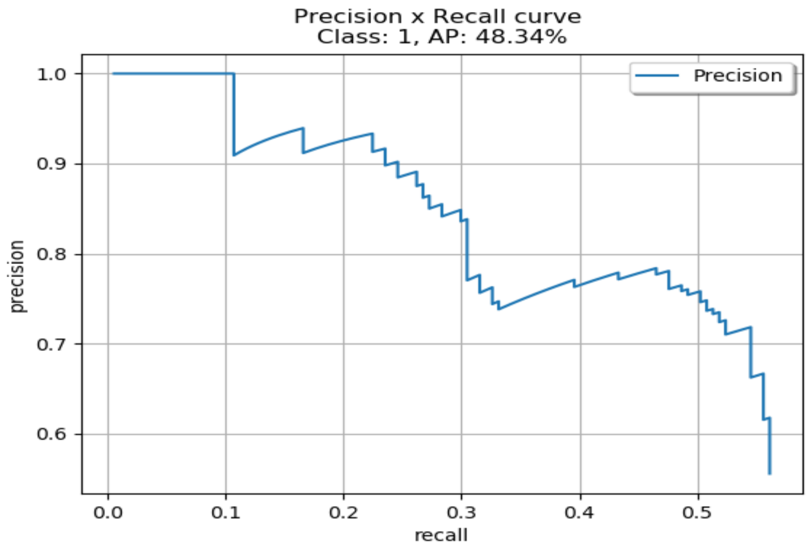


Figure-5: Precision x Recall Curve for class Black-Piece on test set

Conclusion

We don't believe that our results are conclusive. We feel that our results were heavily affected by the dataset of choice and shortcomings in our implementation of the Yolo Framework. A much better comparison can be achieved by experimenting on multiple datasets. Furthermore, we admit that by using the original paper's[1] authors' own code repository, this experiment could have yielded better insights. We recognize this as being bad management and a misjudgement of the problems complexity on our part.

References

1. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

2. J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 6517-6525, doi: 10.1109/CVPR.2017.690.

3. R. Padilla, S. L. Netto and E. A. B. da Silva, "A Survey on Performance Metrics for Object-Detection Algorithms," 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), Niterói, Brazil, 2020, pp. 237-242, doi: 10.1109/IWSSIP48289.2020.9145130.