Multi-Object Detection

Aiman Jabaren
Electrical and Computer Engineering
PID:A53286046
ajabren@ucsd.edu

Hayk Hovhannisyan
Electrical and Computer Engineering
PID: A12466074
hhovhann@ucsd.edu

Aditi Tyagi
Electrical and Computer Engineering
PID: A53279284
adtyagi@ucsd.edu

Khushboo Agrawal
Electrical and Computer Engineering
PID: A53271205
khagrawa@ucsd.edu

Team Information

Team Name: Error 404: Team Not found (all PIDs given above)
Github Link: https://github.com/agrawal-khushboo/Multi-object-detection

Abstract

This paper discusses the implementation of the YOLOv2 algorithm for multi-object detection on the images. The 23 layer YOLOv2 network is trained on the PASCAL VOC'12 dataset. The following paper discusses in detail the training, validation and evaluation techniques followed by the discussion on the challenges and successful methods. For this project we have achieved a mean average accuracy of 11.48 percent over 20 classes.

1 Introduction

Object detection is a computer vision and image processing related problem that incorporates detecting instances of semantic objects of a particular class of objects. Object detection has various applications in many areas and is still a wildly growing research field. To pursue object detection, there are various methodologies. But most of them fall into either machine learning or deep learning based approaches. For our project, we will be focusing on deep learning technique based on convolutional neural network. Particularly, we will be implementing You Only Look Once (YOLO)v2 algorithm for this project. YOLO algorithm trains and optimizes the performance simultaneously. The reason we chose YOLOv2 is because it is extremely fast. Unlike sliding window and region proposal methods, the algorithm works around the entire image to make predictions. And lastly, it learns generalized representations of objects. Even when computer vision looks at deeper, larger networks, better performances are often revolve around blending the network together. This might increase the complexity of the system. For the same reason, we chose YOLOv2, which is a more accurate system with faster processing time. Instead of scaling up the network, we simplify it and make the representation easier to learn. The following report is classified into different sections which provides the detailed implementation. For this project we have achieved a mean average accuracy of 11.48 percent over 20 classes.

Conference on Neural Information Processing Systems (NeurIPS), .

1.1 Targeted Tasks

In the following project, we aim to implement the YOLOv2 algorithm on the PASCAL VOC'12 dataset and detect object in the images and evaluate the results on the test set. The detailed implementation overview is given in the following report.

1.2 Challenges

During the project, we faced some of the challenges mentioned below:

- The loss function, YOLOv2 algorithm uses is difficult to interpret and we found difficult using it, since the loss would not work whenever we loaded the checkpoint.
- For the threshold values we choose, we find that we get multiple bounding box around the same object. Therefore choosing the optimal threshold values is important. We have discussed that in the following report.
- Since the dataset is imbalanced, we found it difficult to train the model unbiased to all the classes. As the recall we get is not evenly distributed hence we find it difficult to increase the overall mean average precision (mAP). We have discussed this in the later part of the report.

2 Description of the method

This part of the report provides a detail description of the methodology used to implement the following algorithm.

2.1 Dataset

For this project we have used the PASCAL Visual Object Classification (PASCAL VOC 2012) dataset. This is a well-known dataset for object detection, classification, segmentation of objects and so on. There are around 10,000 images for training and validation containing bounding boxes with objects. Although, the PASCAL VOC 2012 dataset contains only 20 categories, it is still considered as a reference dataset in the object detection problem. The training dataset consists of set number of images that has an annotation file giving the bounding box and object class label for each object in one of the 20 class labels. For the implementation of the YOLO algortihm we have split the data into 2 categories namely: training, validation which are 90, 10 percent respectively of the entire dataset. For the test data we downloaded the dataset from (4). The use of the respective dataset are listed below:

- Training dataset: used for the training of the network.
- Validation dataset: used for the finding the best threshold value of our prediction.
- Test set: used for the evaluation of the network.

2.2 YOLOv2 Algorithm

The YOLOv2 Algorithm is implemented in the following steps. Figure 1 shows the picture model (1; 2)

- Based on the labels of the bounding box, finding the best suited anchor box number and dimensions.
- Perform input and output encoding by reshaping the image and diving them in 13*13 grids and assigning the anchor boxes
- Initialize the model and pre-trained weights. Pass the encoded data to the the model for training and evaluate on the test data

2.2.1 Anchor Box

The first part of the YOLOv2 algorithm focuses on determining the anchor boxes width and height based on the training data set bounding boxes and their IoU values. To determine the anchor box

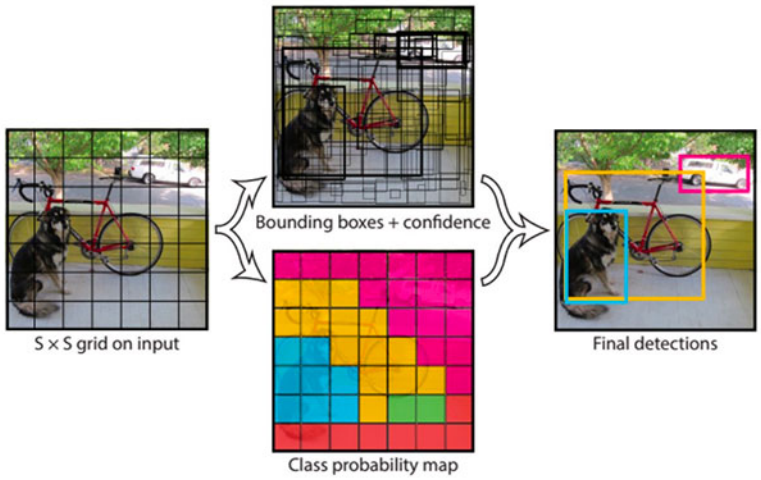


Figure 1: YOLOv2 Model (1)

dimensions, we have used K-means clustering. For the YOLO algorithm, there is only one feature map. The bounding boxes for objects with the lower left and upper right x and y coordinates. However, for clustering we only need the normalized width and height of the box. It is really important to choose the right metric when applying K means. Hence, we choose the Intersection Over Union, also called the Jaccard index, metric (2). If there are 2 boxes b1 and b2, the following equation can be used to get the IoU.

$$J(b1, b2) = \frac{min(w1, w2).min(h1, h2)}{w1h1 + w2h2 - min(w1, w2).min(h1, h2)} \tag{1}$$

The bounding boxes used for the anchor boxes have width and height as the important parameters. These 2 parameters determine the IoU area.

$$IoU = \frac{intersection}{union - intersection} \tag{2}$$

$$intersection = Min(w1, w2)Min(h1, h2) \tag{3}$$

$$union = w1h1 + w2h2 \tag{4}$$

Next, the K means clustering has 2 steps that we follow to set the number of clusters and to initialize the cluster centers.

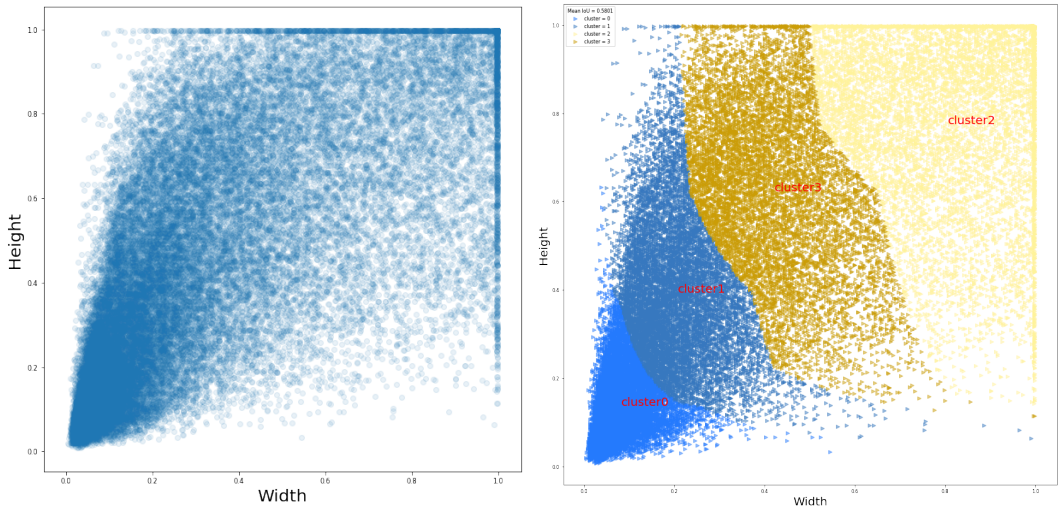
- The allocation of each item to the closest cluster center.
- The calculation of the cluster centers as means of all the cases in the clusters.

We repeat the process till the 2 consecutive iterations result in the same cluster center. We had to make a decision on how many total number of anchor boxes needed to be created. And to make the same decision, we ran the k means for k values from 2 to 11. With more number of clusters, the IoU mean increases, leading IoU to be 1 at the end.
Figure 2 shows the Normalized width and height of the data and the clustering performed on it. The high level idea of the process is that the true number of clusters can be obtained when the increase in the mean IoU slope is large. Hence, we decided to choose 4 anchor boxes.

Table 1 shows the the anchor boxes and their width and heights chosen for further implementation.

2.2.2 Input/Output Encoding

To begin the architecture model of the system, we have a data set reader file that does the input and output encoding. For the input encoding, the image is read and then re-sized. Based on that, the



(a) Normalized width vs height from labeled data (b) Optimal clustering by k-means according IOU

Figure 2: K means clustering for finding the optimal anchor parameters

Anchor Boxes		
Anchor Box	Width	Height
1	1.07709888	1.78171903
2	2.71054693	5.12469308
3	10.47181473	10.09646365
4	5.48531347	8.11011331

Table 1: Anchor box width and heights

output's minimum and maximum x, y coordinates are also re-sized. The ImageReader class does the encoding of passed image parameter. Each object in the training image is assigned to a truth anchor box. The number of anchor boxes used in the project are specified to be 4 and each anchor box has a specific shape given in Table 1.

Next, we have the BestAnchorBoxFinder class that defines the best anchor boxes out of the numerous boxes created using IoU and find modules. The algorithm uses the object's midpoint and the anchor box with the highest IoU.

The YOLOv2 algorithm divides the image into 13*13 grid cells and assigns image classification and localization algorithms in the grid cells. For the encoding purposes, we re scale the bounding box parameters to match the scaling factor of the grid cell parameters.

2.2.3 SimpleBatchGenarator

$SimpleBatchGenerator$ Class is responsible to generate input to our model in the batches. Our model takes 3 inputs so then batch-generator is responsible for providing these inputs. The names of these inputs are given as: **x_batch**, **y_batch**,and **b_batch**

1. **x_batch:** A tensor of $[BatchSize, Image\_H, Image\_W, Channels]$ which is our standard input for an image.
2. **y_batch:** A tensor of $[BatchSize, Grid\_H, Grid\_W, AnchorBoxes, BOX + 1 + Classes]$ which second input that is .This tensor is initialized following way, for each Image i in a batch, $y\_batch[i, :, :, :]$, we loop through all the objects in that images, take the coordinates x_{min}, x_{max},y_{min}, y_{max} and calculate normalized $Center_x$ and $Center_y$ according to:

$$Center_x = 0.5(x_{min} + x_{max}) * \frac{Grid\_W}{Image\_W}$$

$$Center_y = 0.5(y_{min} + y_{max}) * \frac{Grid\_H}{Image\_H}$$

we then ensure that the object coordinates are now within a specific grid by

$$grid_x = \lceil Center_x \rceil$$
$$grid_y = \lceil Center_y \rceil$$

then we calculate object $Center_h$ and $Center_w$ by

$$Center_w = 0.5(x_{max} - x_{min}) * \frac{Grid\_W}{Image\_W}$$
$$Center_h = 0.5(y_{max} - y_{min}) * \frac{Grid\_H}{Image\_H}$$

Using the $Center_w$ $Center_h$ we calculate which one of our anchor boxes is the closest to it by using IOU metric and save it as a number from 1 to 4 in variable called $best_anchor$. This is accomplished with only $Center_w$, $Center_h$ as the box is assumed to be in the corner (i.e $x_{min} = 0$ and $x_{max} = 0$). Putting all these together, $best_anchor$, $grid_x$ and $grid_y$ we then take the our variable

$$y\_batch[i, grid_x, grid_y, best_anchor, 0:4] = [Center_x, Center_y, Center_w, Center_h]$$

then we initialize our confidence for this object to be 1 ,

$$y\_batch[i, grid_x, grid_y, best_anchor, 4] = 1$$

and finally we set our class value using onehot encoding i.e $e = (0, 0, 1, ...0)$

$$y\_batch[i, grid_x, grid_y, best_anchor, 5:5+Classes] = e$$

3. **b_batch:**: this is our last input, which is only passed though our model and is used only for the loss function which simple just takes image count in the batch and object location.

$$b_batch = [i, 0, 0, 0, Object,] = [Center_x, Center_y, Center_w, Center_h]$$

Name	Value	Description
$BatchSize$	45	Number of images per batch
$Image\_H$	416	Image Height in pixels
$Image\_W$	416	Image Width in pixels
$Grid\_H$	13	Number of Horizontal Grids
$Image\_W$	13	Number of Vertical Grids
$AnchorBoxes$	4	Number of Anchor boxes
BOX	4	Normalized Box dimension
$Classes$	20	Number of Object Classes our data set

Table 2: Variable description

The TRUE_BOX_BUFFER helps create the frame with the biggest number of objects can detect all objects, which is beneficial to calculate loss in later parts. The simpleBatchGenerator requires config for the instantiation and it contains various hyperparameters. Out of the list of the hyperparameters shown in Table2, Anchor bounding boxes, which have their width and height in the range of 0 to 1, are adjusted to match the scale of the grid cell scales. Once that is done, the objects detected in the grid cells are assigned one of the anchor boxes out of the 4 described before. Figure **??** are some of the tested examples of the anchor boxes with their box coordinates labelled on the images.

2.2.4 Model

Once we have the 4 anchor boxes and the shape of each box as defined in the previous parts, we define the labels of the 20 classes for the PASCAL VOC 2012 dataset. Compared to the loss and the encoding of the input and output of the YOLOv2 algorithm, the architecture of the model is relatively simpler. In a nutshell, the model is a repeated stack of convolutions, batch normalizations and RELU layers until the image shape is reduced to the grid cell size. The figure 4 is simple overview of the model. (5)

After initializing the above model, we load the pre-trained weights (1) and initialize the first 22 layers of the network before we start training the model and randomly initialize the weights and bias of the last layer.

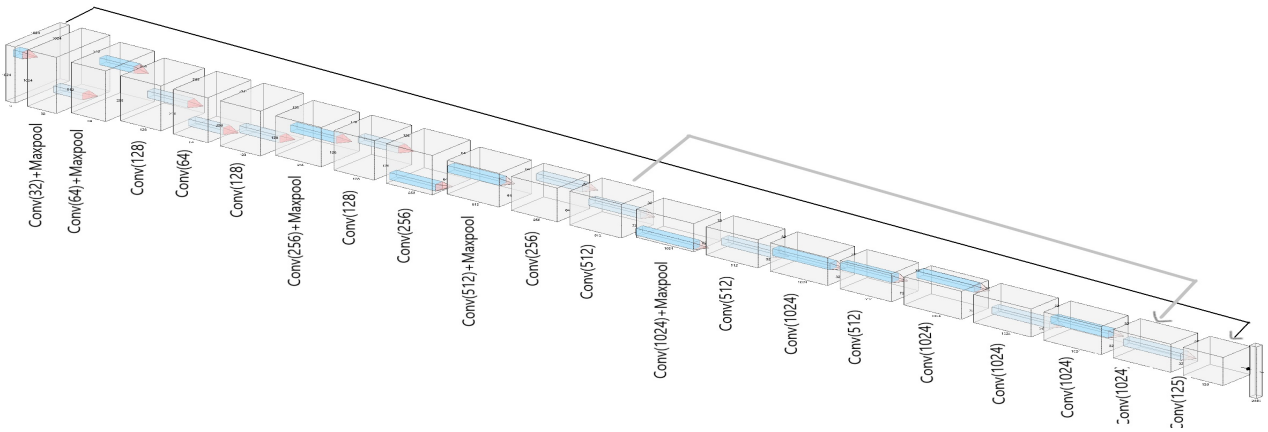


Figure 3: Model of YOLOv2 Implementation

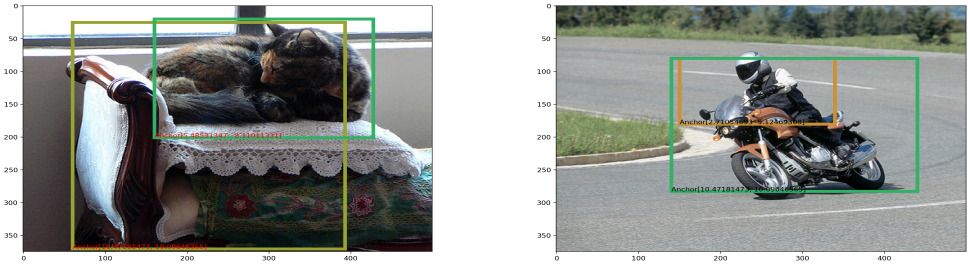


Figure 4: Examples of Anchor Boxes by evalutaion

2.2.5 Loss

The loss of the the algorithm is evaluated in the manner given below (3):
Then the loss corresponding to (grid cell, anchor box) pair = (i,j) is calculated as:

$$loss_{i,j} = loss_{i,j}^{xywh} + loss_{i,j}^{p} + loss_{i,j}^{c}$$

$$loss_{i,j}^{xywh} = \frac{\lambda_{coord}}{N_{Lobj}} \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{i,j}^{obj} [(x_{i,j} - \hat{x_{i,j}})^2 + (y_{i,j} - \hat{y_{i,j}})^2 +$$

$$(\sqrt{w_{i,j}} - \sqrt{\hat{w_{i,j}}})^2 + (\sqrt{h_{i,j}} - \sqrt{\hat{h_{i,j}}})^2]$$

$$loss_{i,j}^{p} = -\frac{\lambda_{coord}}{N_{Lobj}} \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{i,j}^{obj} \sum_{c \in class} p_{i,j}^{c} log(p_{i,j}^{c})$$

$$loss_{i,j}^{c} = \frac{\lambda_{coord}}{N_{Lobj}} \sum_{i=0}^{s^2} (IoU_{prediction_{ij}}^{ground_truth_{ij}} - \hat{c_{ij}})^2 + \frac{\lambda_{noobj}}{N^{conf}} \sum_{i=0}^{s^2} (0 - \hat{c_{ij}})^2$$

where:

- $N_{L^{obj}} = \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{i,j}^{obj}$

- $N^{conf} = \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{i,j}^{obj} + L_{i,j}^{noobj}(1 - L_{ij}^{obj})$
- $prediction_{i,j} = (\hat{x_{ij}}, \hat{y_{ij}}, \hat{w_{ij}}, \hat{h_{ij}})$
- $groung_truth_{i,j} = (x_{ij}, y_{ij}, w_{ij}, h_{ij})$
- $\lambda_{coord}, \lambda_{class}, \lambda_{noobj}$ are scalars to weight each loss function

Here L_{ij}^{noobj} and L_{ij}^{obj} are 0/1 indicator function such that:

$$L_{ij}^{obj} \begin{cases} 1, & \text{if } C_{ij} = 1 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

$$L_{ij}^{noobj} \begin{cases} 1, & \text{if } max_{ij} IoU_{prediction_{ij}}^{ground_truth_{ij}} < 0.6 < C_{ij} = 0 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

3 Training

After defining the above functions, for which you will get the link above, we trained the network by choosing the following parameters given in the table 3, and the loss defined above. Figure 5 shows the plot of the loss.

Training Parameters	Values
learning rate	0.5e-4
decay rate	0
batch size	32
momentum	0.9, .99
epoch	50
epsilon	1e-08
lamda-no-object	1
lamda-object	5
lamda-class	1
lamda-class	1

Table 3: Training parameters

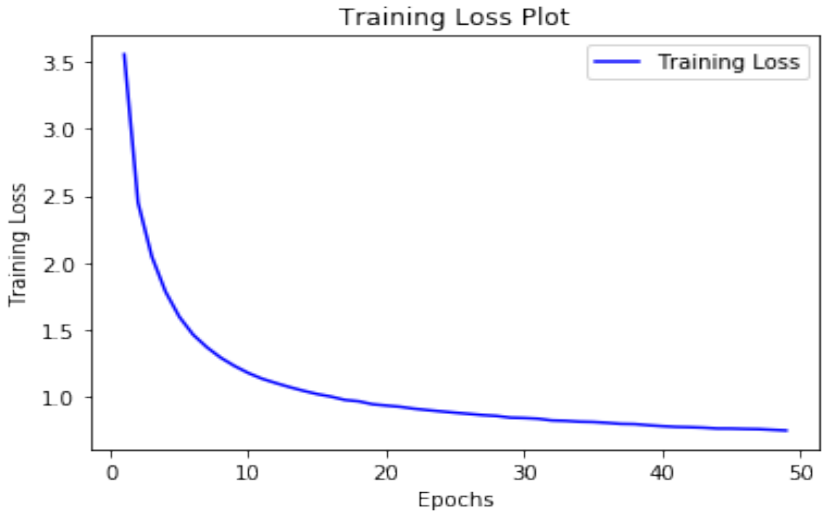


Figure 5: Training Loss Plot

4 Validation

Since our loss function was performing well and was continuously decreasing, we choose the validation set to find the hyper-parameters which are the threshold values of the IoU and the confidence. We have chosen roughly 10 percent of the dataset.

Before discussing the precision metrics, we will comment on the displayed images' performance. As seen, the detection is very sensitive to the given thresholds. As mentioned previously, in Yolo algorithm, there are two threshold that play an important role in deciding whether a suspected object should be classified as a detected object. The two thresholds are: the confidence threshold which indicates the probability that a box has been classified correctly. Intuitively, we would like this threshold to be high enough to eliminate False Positives (missclassified boxes), but not too high that it might miss objects of interests. Similarly, there is an IoU threshold which eliminates all detected objects that have a lower IoU overlap; This threshold serves as an indicator that a True Positive (correctly detected object) and the ground truth boxes should have a certain overlap. Again, a suitable IoU threshold should serve as an appropriate indicator whether the predicted box (object) matches the ground not only in label, but also in location. For this purpose, a short evaluation experiment was implemented on the validation set in order to evaluate the different threshold combinations' performance as shown in the heatmap given in figure 6.To generate the heatmap below, we created a list of IoU and confidence thresholds. Different Precision and Recall values were generated for different threshold values. The Average precision was then plotted and a threshold value of 0.6 was used to detect objects in the images.

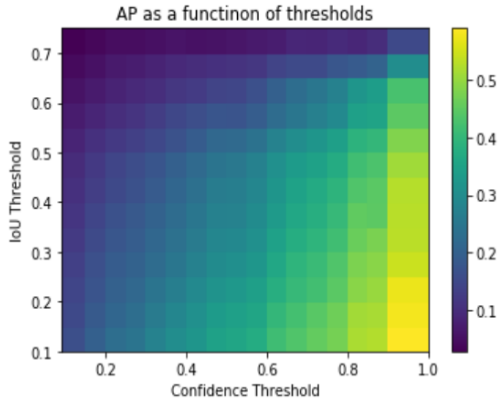


Figure 6: IoU vs. Confidence Threshold HeatMap with Average Precision Values

5 Evaluation and Results

5.1 Evaluation

For performance evaluation, we investigate the detected objects accuracy. As discussed, the lower the accuracy threshold, the more detected objects because objects with lower confidence will be included too. The lower IoU threshold is, the more detected objects too due to including boxes that are farther from the ground truth. Ideally, we would like the system to detect only the desired objects. But since the accuracy of the detection varies in location and accuracy according to label and image, we desire optimal thresholds that would serve best for the average test image. For this purpose, a short evaluation experiment was implemented on the validation set in order to evaluate the different threshold combinations' performance.

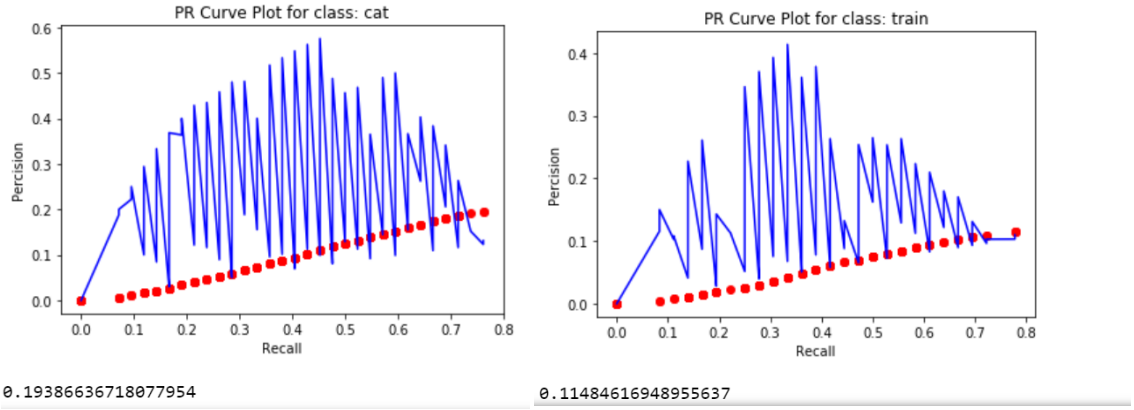
The evaluation metric for YOLOv2 is discussed in the section below. Average Precision (AP) is a popular metric used to measure the accuracy of the object detectors. To do the same, we compute 2 main parameters: Precision and Recall. Precision is the value of how accurate the predictions are while recall is the measure of how good we find all our positives. The measure of finding positives is taken including both true and false positives. In the earlier sections of the project, we save the True Positives, True Negatives and False positives successfully by comparing the predictions to the

IoU and confidence thresholds. The following equations are used to calculate precision, recall and Average Precisions for each class (labels). (6)

$$\begin{aligned} Precision &= \frac{TP}{TP+FP} & TP &= TruePositive \\ Recall &= \frac{TP}{TP+FN} & FP &= FalsePositive \\ & & TN &= TrueNegative \\ & & FN &= FalseNegative \end{aligned} \tag{7}$$

$$\begin{aligned} AveragePrecision(AP) &= \int_0^1 p(r)dr \\ MeanAveragePrecision(mAP) &= \frac{\sum_{class} AP}{Numberofclasses} \end{aligned} \tag{8}$$

For each class, we evaluate the AP based on the confidence and the IoU threshold values and plot their Recall and Precision correlation as shown in the figure 7. The graph shows the precision and recall for different thresholds and the AP given by the red dots. Here we have just shown the graphs for 2 classes with the AP mentioned in the left below. After evaluating the AP we evaluate mAP using eq 8. The mAP calculated is 11.48 percent.



(a) PR plot of Cat Class

(b) PR plot of Train Class

Figure 7: Precision-Recall Plot for Average Precision

6 Discussion

In this part of the report, we aim to discuss the highlights of what we learnt, challenges we faced, and the success and failures we encountered. We believe this is the most crucial part of the report and the important challenges and findings are discussed further.

- While implementing the YOLOv2 algorithm, we learnt new methods for encoding the input and output images, implementing K-means using IOU metric using training annotations to find the best anchor boxes and the YOLOv2 algorithm.
- While defining the loss given by Keras library, we had complications in starting the training of the model from the checkpoint. After researching on the topic, we found that the input hack layer breaks after the kernel shuts. This was one of the major challenge. The way we tackled the problem was by looking for one of the custom loss functions (3) and the model improved immensely.
- Since our model training loss was decreasing consistently we chose to eliminate early stopping and use the validation set wisely for finding the threshold values that would further be used for evaluation.

- Another findings were that visually the bounding boxes gave good results, but once the model was run, we observed multiple bounding boxes detecting the same object. Some of the boxes was shifted with an offset, caused due to similar IoU values. After minor tweakings of the threshold values, we found that the threshold values are very important for this problem and we need to optimize the values for the model to work efficiently.
- Once we optimized the thresholds by evaluating the model on the validation set, we plotted the Precision as a function of IoU and Confidence thresholds. The plot gives us the AP per class as a function of the recall. Optimal threshold combinations would give us highest Average Precision on the heatmap on investigating the heatmap.
- For the evaluation of YOLOv2 model, we used a commonly used Average Precision metric. We achieved a mAP value of 11.48 percent for all the classes. We believe that this values is good but maybe not the best due to the following reasons: The 20 classes of the training data set is unbalanced as we see that the person class has the highest number of examples whereas train and bus have the lowest examples. Second reason may be that recall of the function for all classes is not similar that is between 0 and 1 or to say unevenly distributed over the classes.
- To eliminate this, for future works of the project, the following implementations can be incorporated to get a higher mAP value. Firstly, we can implement a suppression function that would smooth the Precision vs Recall plot and give higher area under the graph.
- Secondly, the range of the achieved recall values is less than [0,1]. In future works, we can iterate over more threshold values that provide a Precision-Recall coordinates with a recall covering the whole range. This increase in the recall range will give us a higher area under the graph and increase the calculated mAP values.
- In order to get a better and more reflecting PR graph, we could smooth the PR graph and denoise it by interpolating the max points in small intervals.
- Another issue that may be solve in the future is the fact that for certain threshold coordinates, some classes never get predicted and therefore would detect zero objects of that class. This would cause data scarcity and distort the PR graph. In order to solve this, we may add more data or retrain our model.

Acknowledgement

The authors would like to thank Prof. Charles Deledalle for giving us the opportunity to implement deep learning methods through this project and teaching assistant Abhilash Kasarla for his guidance.

References

[1] Joseph Redmon, Santosh Divvala†, Ross Girshick¶, Ali Farhadihttp://pjreddie.com/yolo/. 2016.

[2] Yumi https://fairyonice.github.io

[3] experiencor https://github.com/experiencor/keras-yolo2 ‘

[4] https://pjreddie.com/projects/pascal-voc-dataset-mirror/?fbclid=IwAR1Iw_OdHpdtf94p9awP6-YmlV7sn5zx5-YM6u9MJn1Adtxx8i6DOWCHXHo

[5] github.com/allanzelener/YAD2K

[6] https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173fbclid=IwAR15g6qZSOrxXxDxzG_djc6C_XsOz8mQ1G4IqDqrKtS3DVzARSZgMncqrhA