YOLO Project Report

M Shiva Krishna Reddy (CS13B051), Y Sasi Kiran (CS13B055)

May 4, 2017

1 Introduction

Object detection in visual systems is becoming important in many applications with emphasis on accurate and reliable real time systems. R-CNN [11] , Fast R-CNN [3] and Faster R-CNN [4] are some of the models which try to address the problem of object detection. But these models only reach a frame rate of 7 frames per second. YOLO [16] is a step in the direction of achieving real time object detection with frame rate of about 45 fps, while still achieving reasonable accuracy on Pascal 2007-VOC dataset [10]. Several variants of YOLO have been proposed [15] which improve upon the accuracy while maintaining the frame rate. In this work we experiment with YOLO and its variant and observe the influence of different design decisions in the final results.

2 Literature Survey

Convolutional Neural Networks have become popular due to the large availability of data and the advent of accelerated hardware for computation. CNNs have become popular in tasks related to Computer Vision after their application in the winning submission of IMAGENET Challenge [7]. Recently CNNs have been applied to various fields of computer Vision like Object Detection [11] [3] [4], Object Recognition [7] [14] [6], Semantic Segmentation [9] [5], Activity Recognition [8] [13], etc.

Object Detection is the task of identifying all the objects in the given image and localizing them using a bounding box. The work on Object Detection prior to Deep Learning poses the localization as a regression problem. Handcrafted features are extracted from the image and a regressor is learnt which outputs the bounding box co-ordinates in the image. These have achieved a very poor mAP of 30.8% on PASCAL VOC challenge as observed by [11].

RCNN [11] uses convolutional neural networks to solve the problem of Object Detection. RCNN first extracts region proposals using Selective search, extracts the CNN features for each proposal and trains a classifier and bounding box regressor for predicting the class of the object as well as its location.

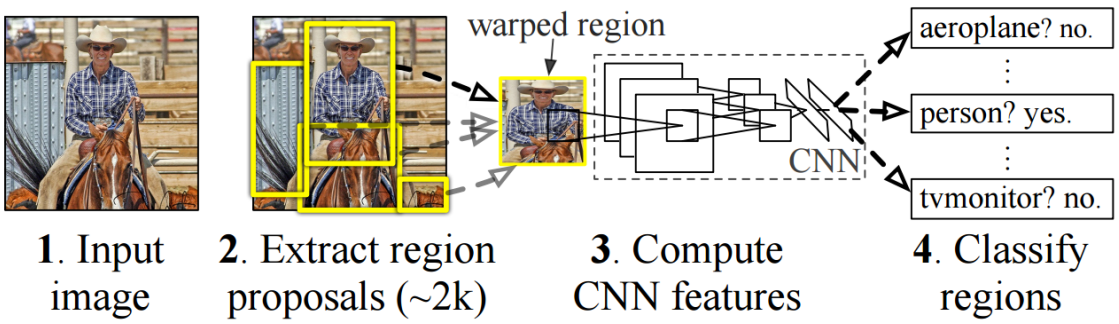

Figure 1: An overall summary of the method used in rcnn for object detection.

RCNN is quite slow in its predictions taking a time of 20sec per each test image. Several variants of RCNN have been proposed which improve the performance of RCNN and have a better

processing time per image. SPPNet [12] and Fast RCNN [3] avoid extracting the features for each region proposal separately and pass the image once through CNN and extract the features corresponding to each proposal from the final feature map. Faster RCNN [4] further improves the performance using a CNN(Region Proposal Network) to extract the region proposals.

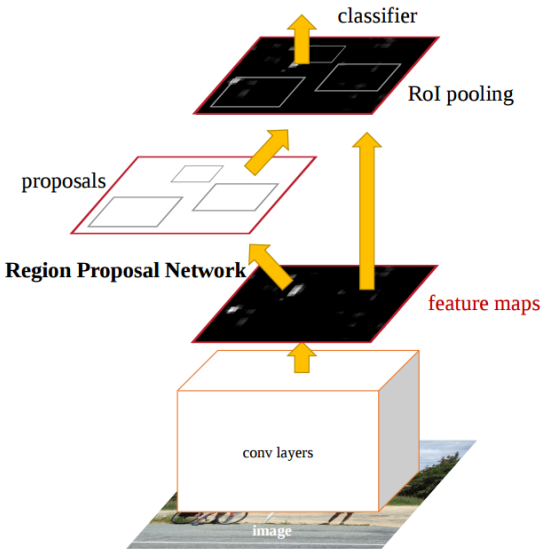

Figure 2: The hierarchical architecture used in Faster RCNN which consists of region proposal network and the classifier.

Though Faster RCNN has a good accuracy on the object Detection task, it still can not be used in real-time systems because of its higher processing time per each image ($140ms/image$). Also Faster RCNN is not a unified model and its training is done in several parts where the region proposal network is trained first and the whole network is trained again.

YOLO [16] proposes an object detection system which works in near real-time with a unified model(single CNN) for Object Recognition. YOLO divides the given image into a regular grid. For each cell in the grid, the CNN is trained to predict bounding boxes and class probabilities. Each bounding box prediction consists of the box co-ordinates (x,y,w,h) and the confidence of the bounding box (c) which models its Intersection over Union (IOU) with ground truth and probability of presence of an object.

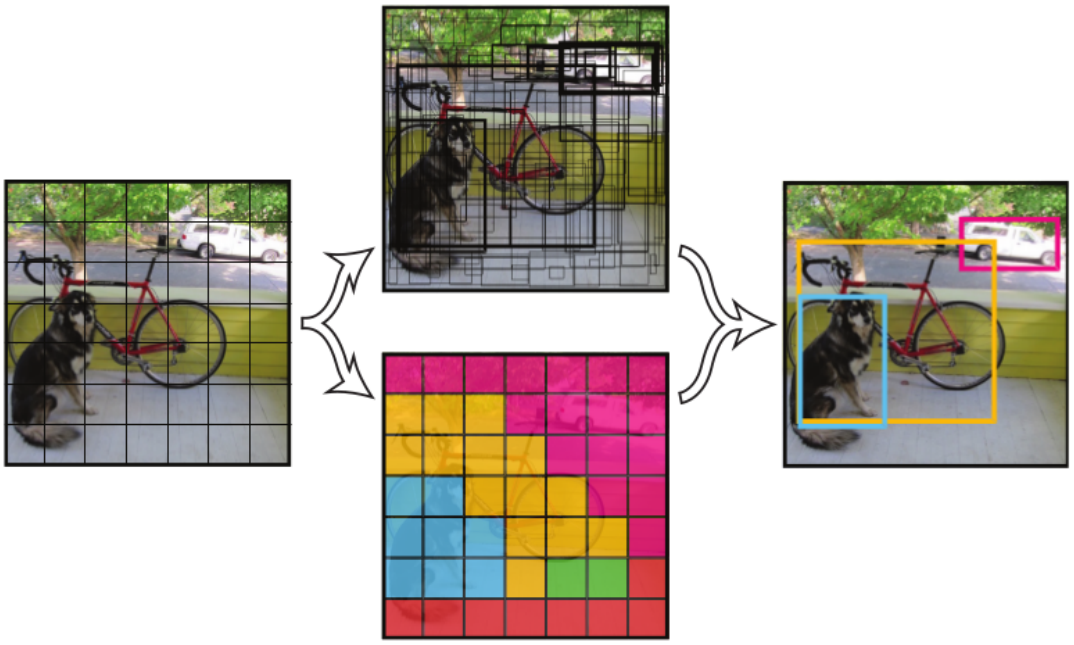

Figure 3: The graphical summary of the method used for bounding box predictions in YOLO.

The predictions are made in the form of a feature vector which is obtained from the last fully connected layer in the neural network. The CNN model used is similar to the the Google-Net model where the inception module consists of only 3×3 convolutions. This architecture is shown in the figure 4.

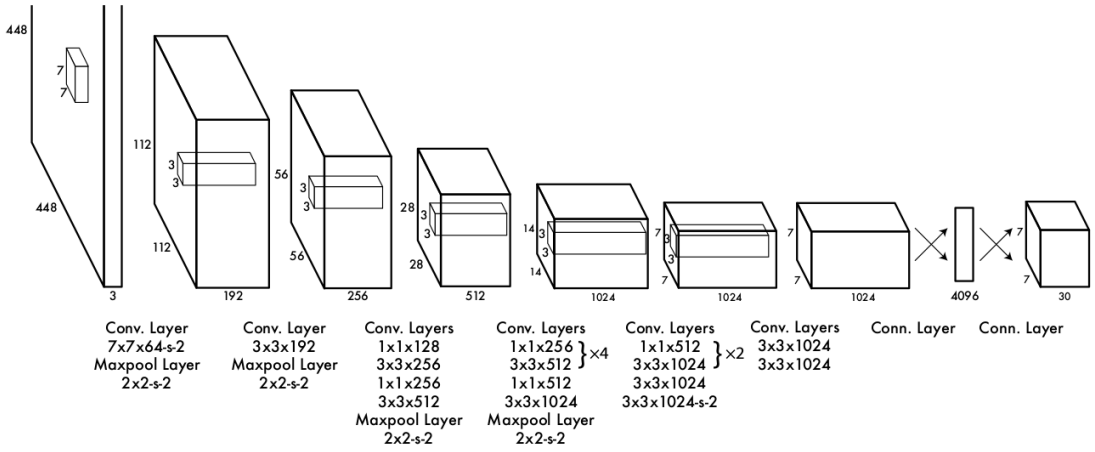

Figure 4: The CNN architecture used in YOLO

YOLO performs the object detection task in real-time with a frame rate of $45 fps$. It achieves $63.4\% mAP$ on PASCAL VOC dataset. Several improvements for YOLO have been proposed which improve the predictions of the model while maintaining its frame rate and the model has been named as YOLO v2 [15].

YOLO v2 improves the performance of the model using several variations. It introduces Batch Normalization [1] after each convolutional layer which helps in faster and better learning. The main variation with the original model is the introduction of anchor boxes in YOLO v2. First,

the most relevant anchor boxes for the data-set are extracted as prior anchor boxes by clustering all the ground truth bounding boxes using K-means clustering with IOU as the distance measure. The cluster centers of K-means are taken as reference anchor boxes.

Then at each location in the grid cell, the changes to these anchor boxes are predicted. This makes the model easily trainable since we only need to predict changes to prior boxes and also results in a significant improvement in performance ($5\% mAP$). Also the model is trained at different resolutions of input image to identify the objects at different scales with a better accuracy.

3 Datasets

We use Pascal 2007-VOC and Pascal 2012-VOC [10] datasets for this project. Training is done using training data from both 2007-VOC and 2012-VOC. Testing is done only on 2007-VOC test data so as to compare the results with many papers including R-CNN based and YOLO which show the results on this dataset.

4 Experiments

Experiments are done with mean average precision : mAP(described in next section) and mean Recall as metrics.

4.1 YOLO

Instead of proposing regions and then detecting objects, YOLO performs an end to end detection of objects as described in the above section. We use the code written in C from the authors of YOLO [2] to train on the VOC data. The authors' code was exported in a framework called Darknet which allows us to make modifications to network while abstracting out the training. Testing code was written in C and partly in python by us. Training is done on a single Titan-X GPU. We were able to achieve a mAP of 61.2 close to the author's claim of 63.4 .

The following experiments are done after studying the work by [15] called YOLO-v2

4.2 Transfer Learning

To have better initializing for the network, we train the network on IMAGE-NET for 10 epochs and use those weights for initializing the network. This improves the mAP to about 64% . The weights after training on Image-net are taken from [2]. This shows the efficiency of transfer learning on object detection.

4.3 Better priors using K-Means

In YOLO anchor boxes are hand crafted with varying width and height. Instead of doing that, one could initialize the anchor boxes with priors learned from the data itself. By treating each anchor box as a point in space, K-Means clusering on the bounding boxes is done to get good shapes for the anchor boxes. Here K=5 gave good results for the authors and they provide the shape of the anchor boxes. We use the same anchor boxes as priors.

4.4 Batch Normalization

Better regularization often results in better results. YOLO uses drop out as regularizer. [15] uses batch normalization. We tried with both batch normalization and drop out which gave poorer results when trained for same number of epochs. Finally we also use only batch normalization.

4.5 Frame rate vs Accuracy

YOLO-v2 removes the FC layer from the network, and predicts K=5 bounding boxes for every location in the output feature map. This makes the model easy to adapt to different input sizes. Also the network itself can be made shallow or deeper depending on the acccuracy needed. We train such a model using a shallower network, to achieve a reasonable mAP of **50.7%** achieving a huge frame rate of **243** fps on the Titan-X GPU. With a deeper network we achieve an mAP of **74.97%** at a frame rate of **76** fps close of YOLO-v2 author's claim of 78% using more advanced techniques such as wordtrees etc.

4.6 Thresholding

In the mAP estimation *Zisserman* [10] proposes an IOU (Intersection over union) of atleast 0.5 to be considered a correct bounding box. For this IOU we look at various thresholds on the confidence of each box, to be considered as a predicted box. The results are shown in Table-1

Threshold	**mean Precision(%)**	**mean Recall(%)**
0.1	62.3	84.0
0.2	72.39	82.05
0.25	74.97	81.4
0.5	84.0	74.7

Table 1:

4.7 Evaluation Metrics

We consider only bounding boxes with IOU greater than 0.5 with any of the ground-truth boxes as correctly retrieved matches as proposed by Zisserman. Now, we make a matching between predicted and ground-truth boxes for the test data with prioirity to class label being the same and then the IOU match. After this mathcing is done, we find precision and recall. The average over the precision values across images averaged over discrete (K=11) recall levels is called Mean Average Precision or mAP and is reflective of the area under the ROC curve. This is used as the evaulation metric along with mean Recall.

4.8 Results at a glance

Method	**mAP %**	**mAP (Author's Claim)**
YOLO	61.2	63.4
YOLO with Tranfer learning	64.0	–
Better priors and regularization	**74.97**	–
Tiny-YOLO (243fps)	50.7	–
YOLO-v2 at (76fps)	**74.97**	78.6 at (70fps)
Thresholding of 0.5	**84.0***	–

Table 2: *: Mean Precision at loss of 6% Recall, not mAP

5 Conclusions and Future work

We experiment with methods descibed in [15] and show results close to stated results. Thus YOLO-v2 improves over the existing object detection framework with several innovations along with real time results using above mentioned techniques. Also due to unified structure YOLO is much faster to train using a single GPU. We conjecture that a model like a convolutional pose machine where results are iteratively improved will give better performance at slight trade off with detection time. We intend to try this out in near future.

6 Some Results

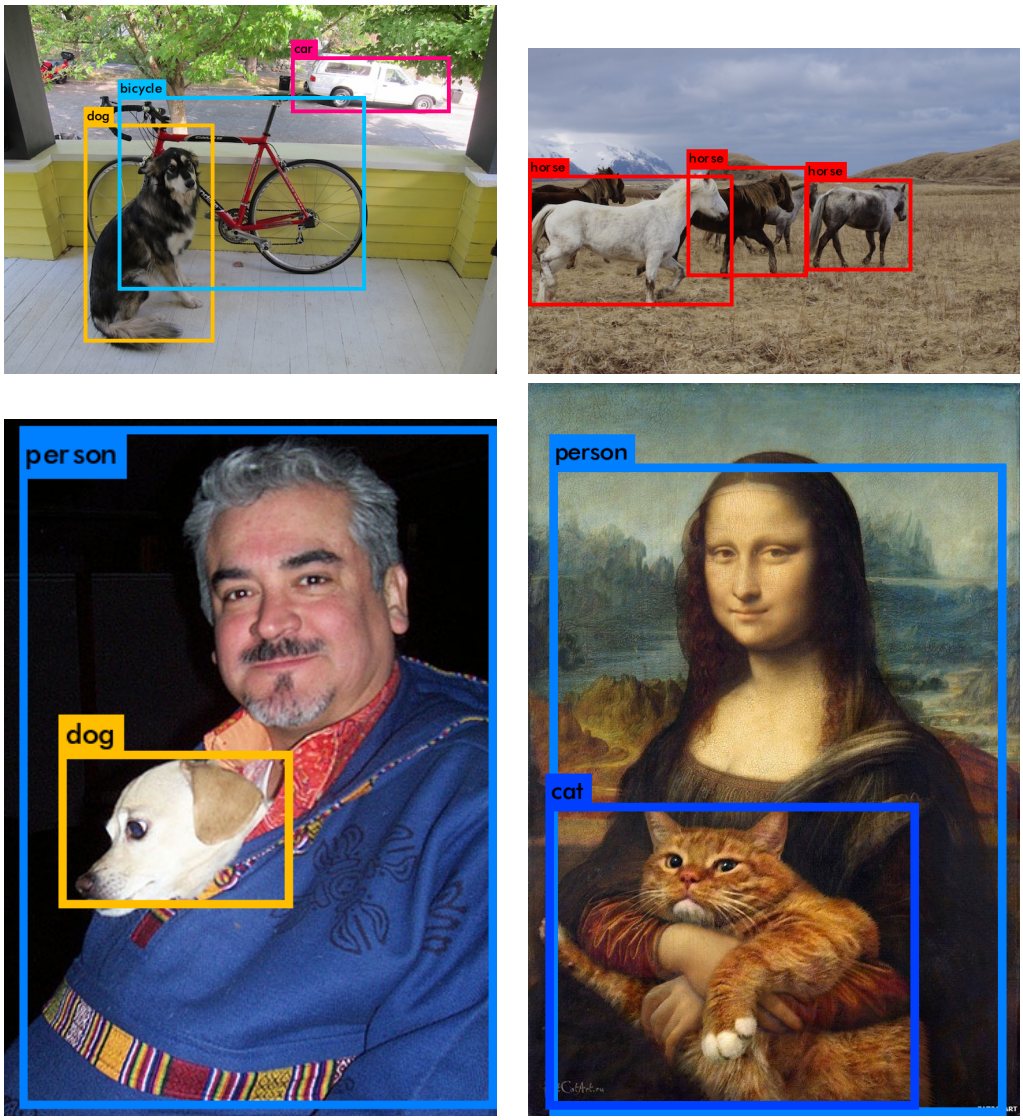

Figure 5: Good Resolution Invariance on Test images

References

[1] Batch normalization: Accelerating deep network training by reducing internal covariate shift. https://arxiv.org/pdf/1502.03167.pdf.

[2] Darknet. https://github.com/pjreddie/darknet.

[3] Fast r-cnn. https://arxiv.org/pdf/1504.08083.pdf.

[4] Faster r-cnn: Towards real-time object detection with region proposal networks. https://arxiv.org/pdf/1506.01497.pdf.

[5] Fully convolutional networks for semantic segmentation. https://arxiv.org/pdf/1605.06211.pdf.

[6] Going deeper with convolutions. https://arxiv.org/pdf/1409.4842.pdf.

[7] Imagenet classification with deep convolutional neural networks. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[8] Large scale video classification with convolutional neural networks. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42455.pdf.

[9] Learning deconvolution network for semantic segmentation. https://arxiv.org/pdf/1505.04366.pdf.

[10] The pascal visual object classes (voc) challenge. http://host.robots.ox.ac.uk/pascal/VOC/pubs/everingham10.pdf.

[11] Rich feature hierarchies for accurate object detection and semantic segmentation. https://arxiv.org/pdf/1311.2524.pdf.

[12] Spatial pyramid pooling in deep convolutional networks for visual recognition. https://arxiv.org/pdf/1406.4729.pdf.

[13] Two-stream convolutional networks for action recognition in videos. https://arxiv.org/pdf/1406.2199.pdf.

[14] Very deep convolutional networks for large-scale image recognition. https://arxiv.org/pdf/1409.1556.pdf.

[15] Yolo9000: Better, faster, stronger. https://arxiv.org/pdf/1612.08242.pdf.

[16] You only look once: Unified, real-time object detection. https://arxiv.org/pdf/1506.02640.pdf.