Object Recognition and Ontology Generation for Qualitative Scene Description

Dipika Boro, Shruti Mohanty, Zubin Bhuyan

{dipika_boro, shruti_mohanty, zubin_bhuyan}@student.uml.edu

CS 5450, Computer Science, UMass Lowell

Abstract. *Object detection is an important open problem which has implication in many domains. For our project we have adopted the YOLOv2 for object detection. We implement an object detection framework in Python and train it in the PASCAL VOC dataset. Our system also generates an RDF file describing the bounding box and types of objects detected.*

Keywords: Object detection, CNN, RDFS

1. Introduction

In times where science and technology are trying to understand and replicate humans, object recognition has immerged as a major issue. The ultimate goal of object recognition is to strive to archive the performance of a human eye: to retrieve the information that aren't apparent from a picture, to categorize them into their respective categories, and to organize them in such a way in the memory that the orientation of each object with its surrounding objects remains intact. Object recognition in images and videos is an open problem relevant in several domains such as autonomous driving and assistive robotics. Recent developments in deep neural networks have enabled researchers to design more generalized framework for object recognition and scene understanding. A few of the successful systems have been proposed in recent years, but comparisons are difficult due to different base feature extractors (e.g., VGG, Residual Networks), different default image resolutions, as well as different hardware and software platforms. There is a tradeoff between accuracy and speed while choosing any one method. In our project, we have applied the current state-of-the-art method: YOLOv2 (You Only Look Once version 2). In contrast to other region proposal classification networks (like fast R-CNN [7]) which performs detection on various region proposals and thus end up performing prediction multiple times for various regions in an image, YOLO architecture is more like a fully convolutional neural network and passes the image once through the network and outputs the prediction. The architecture splits the input image into a grid and for each grid, bounding boxes and class probabilities for those bounding boxes are generated. We have used PASCAL VOC 2007 and 2012 to train and test our system.

2. Background

Over the years computer vision research communities have developed and used many different methods for tackling the task of Object Detection. Most of the methods for Object Detection can be broadly classified as either Classical approach or Deep learning approach.

2.1 The Classical approach involves local feature based recognitions, key point based object detectors and descriptors etc. One of the major breakthroughs was a framework developed by Paul Viola and Michael Jones, in 2001 [1], which is called the Viola-Jones Object Detection framework. Another traditional method is using Histogram of Oriented Gradients features and support Vector Machines for classification.

2.2 Deep Learning has been a real game changer in Machine Learning, especially in Computer Vision. In 2012 Alex et.al. [2] used Deep Convolutional Neural Networks for image classification on ImageNet dataset. After that, Deep Neural Networks have been used extensively for the task of Object Detection as well. One of the major advances in using deep learning in object detection was OverFeat [5] developed in 2013.

The following are some of the methods currently noteworthy for the task of Object Detection.

2.2.1 R-CNN

In R-CNN: Region Based Convolutional Neural Networks [6] by Ross Girshick et.al proposed a three-stage approach for Object detection. 1. Extract possible objects using a using a region proposal method. 2. Extract features from each region using a CNN. 3. Classify each region with SVMs.

2.2.2 Fast R-CNN

A more efficient approach based on R-CNN was developed in 2015, called Fast R-CNN [7]. This approach overcomes some of the problems in R-CNN and removes the use of SVMs for classification thus making it pure deep learning approach. Similar to R-CNN, it used Selective Search to generate object proposals, but instead of extracting all of them independently and using SVM classifiers, it applied the CNN on the complete image and then used both Region of Interest (RoI) Pooling on the feature map with a final feed forward network for classification and regression.

2.2.2 YOLO

Shortly after Fast R-CNN was developed, another approach, called YOLO was proposed by Joseph Redmon et.al. [4]. It was an important development, as it was for the first time that Object Detection was carried out in real time. YOLO stands for You Only Look Once. As in the name, this approach looks at the image only once, and then identifies the objects in the image in real time.

There are currently two versions of YOLO- YOLOv1 [4] and YOLOv2 or YOLO9000 [3]. Their approach is discussed in detail in section 3.

3. Approach

For our project we have decided to adopt the approach proposed by Redmon et.al in [3]. Their architecture is popularly known as YOLO9000, which is the second version of the YOLO v1 [4]. This approach is called YOLO (short for "You Only Look Once") because the program needs to look the image only once, unlike other approached where a repurposed image classifier look at an image multiple times through a sliding window.

The YOLO process begins by gridding an image into 13 x 13 cells. Each cell is responsible for predicting 5 bounding boxes enclosing an object. It outputs a confidence score that indicates how certain it is that the predicted bounding box actually encloses an object. For each bounding box, the cell also predicts a *class*. (It gives a probability distribution over all the possible classes.)

The confidence score for the bounding box and the class prediction are combined into one final score that tells us the probability that this bounding box contains a specific type of object. In Fig 3, the thickness is proportional to the confidence score.

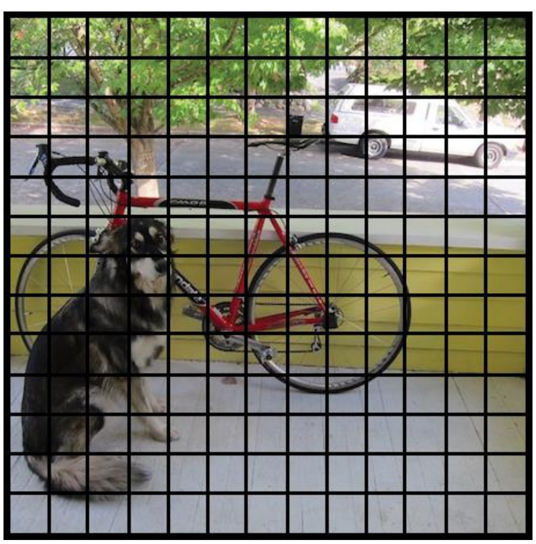


Fig 2 Image divided into 13x13 grid.

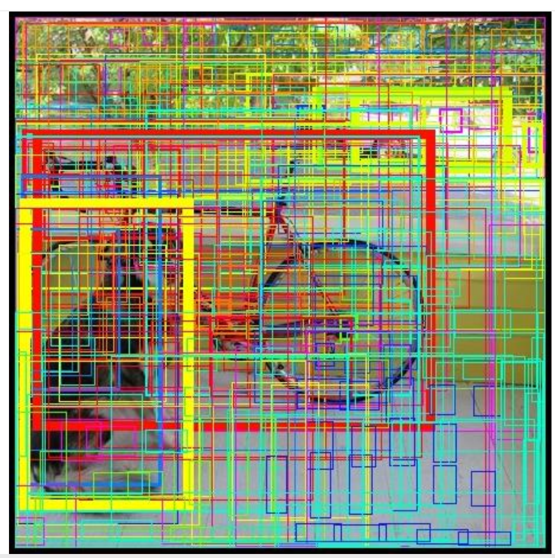


Fig 3 Image with predicted bounding boxes.

There are 169 grid-cells, and each cell can belong up to 5 bounding boxes. Thus, we can have a maximum of 845 bounding boxes. But since most of these boxes have low confidence, we can discard most of them and retain only those which have a score higher than a specified threshold. Fig 4 shows the final output with 3 bounding boxes.

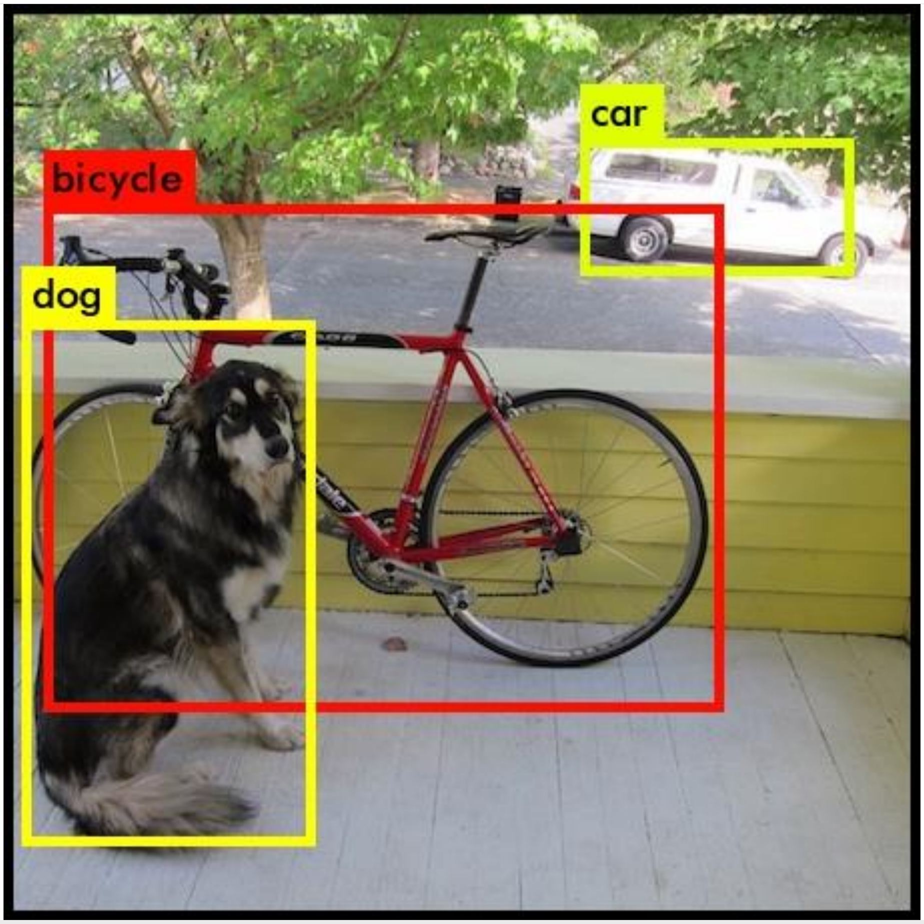


Fig 4 The three bounding boxes with values higher than threshold were retained.

The classification model used is called "Darknet-19" [8]. It has 19 convolutional layer and 5 max-pooling layers. Table 1 shows how the layers are arranged.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Table 1 Layers of the YOLO v2 network.

They train the network on standard ImageNet 1000 class classification dataset for 160 epochs using stochastic gradient decay (learning rate of 0.1). After training for classification, they modify this network for object detection by removing the last convolutional layer and instead adding on three 3 × 3 convolutional layers with 1024 filters each followed by a final 1 × 1 convolutional layer with the number of outputs needed for detection.

Detection Frameworks	Train	mAP	FPS
Fast R-CNN	2007+2012	70.0	0.5
Faster R-CNN VGG-16	2007+2012	73.2	7
Faster R-CNN ResNet	2007+2012	76.4	5
YOLO	2007+2012	63.4	45
SSD300	2007+2012	74.3	46
SSD500	2007+2012	76.8	19
YOLOv2 288×288	2007+2012	69.0	91
YOLOv2 352×352	2007+2012	73.7	81
YOLOv2 416×416	2007+2012	76.8	67
YOLOv2 480×480	2007+2012	77.8	59
YOLOv2 544×544	2007+2012	**78.6**	40

Table 2 Detection on PASCAL VOC 2007.

4. Results

4.1 Dataset

We have used PASCAL VOC 2007 and 2012 to train and test our system. The dataset has a total of 20 class objects and corresponding segmentations for each object. In PASCAL Visual Object Classes dataset, there are 20 classes categories:

	train		val		trainval	
	img	**obj**	**img**	**obj**	**img**	**obj**
Aeroplane	112	151	126	155	238	306
Bicycle	116	176	127	177	243	353
Bird	180	243	150	243	330	486
Boat	81	140	100	150	181	290
Bottle	139	253	105	252	244	505
Bus	97	115	89	114	186	229
Car	376	625	337	625	713	1250
Cat	163	186	174	190	337	376
Chair	224	400	221	398	445	798
Cow	69	136	72	123	141	259
Diningtable	97	103	103	112	200	215
Dog	203	253	218	257	421	510
Horse	139	182	148	180	287	362
Motorbike	120	167	125	172	245	339
Person	1025	2358	983	2332	2008	4690
Pottedplant	133	248	112	266	245	514
Sheep	48	130	48	127	96	257
Sofa	111	124	118	124	229	248
Train	127	145	134	152	261	297
Tvmonitor	128	166	128	158	256	324
Total	2501	6301	2510	6307	5011	12608

Table 3 PASCAL VOC 2007

	train		val		trainval	
	img	**obj**	**img**	**obj**	**img**	**obj**
Aeroplane	327	432	343	433	670	865
Bicycle	268	353	284	358	552	711
Bird	395	560	370	559	765	1119
Boat	260	426	248	424	508	850
Bottle	365	629	341	630	706	1259
Bus	213	292	208	301	421	593
Car	590	1013	571	1004	1161	2017
Cat	539	605	541	612	1080	1217
Chair	566	1178	553	1176	1119	2354
Cow	151	290	152	298	303	588
Diningtable	269	304	269	305	538	609
Dog	632	756	654	759	1286	1515
Horse	237	350	245	360	482	710
Motorbike	265	357	261	356	526	713
Person	1994	4194	2093	4372	4087	8566
Pottedplant	269	484	258	489	527	973
Sheep	171	400	154	413	325	813
Sofa	257	281	250	285	507	566
Train	273	313	271	315	544	628
Tvmonitor	290	392	285	392	575	784
Total	5717	13609	5823	13841	11540	27450

Table 4 PASCAL VOC 2012

4.2 Implementation

We implemented the YOLO v2 architecture in Python using the PyTorch library. We have used libraries and modules for additional functionalities which are mentioned in the footnotes. Python 2.7, PyTorch v0.4[1] and CUDA 8 is required for the execution of the program. We trained our model for 160 epochs on the combined training sets of PASCAL VOC[2] 2007 and 2012.

After the detection of objects in a test image, an RDF file is generated which stores the information like type of objects detected and its bounding box coordinates. Since a bounding box is always a rectangle, we store only the diagonally opposite points. This information can be used by other applications to further analyze the scene.

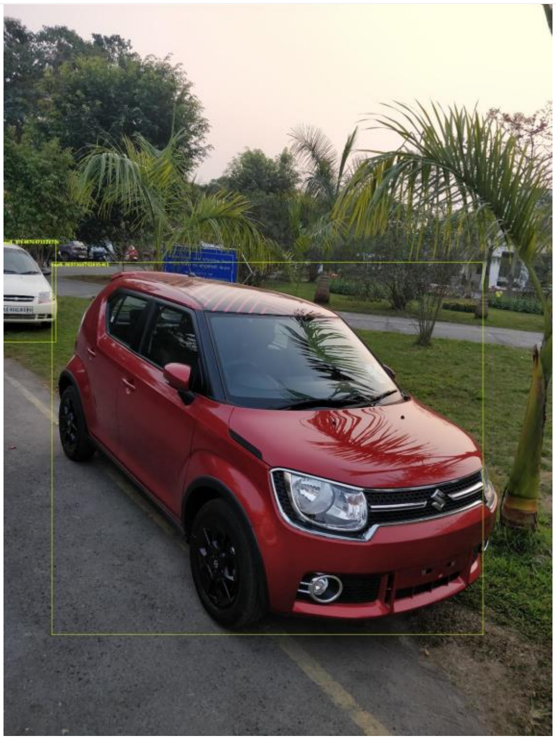
We have also incorporated the Darknet for Python[3] so that it is possible to use the weights of darknet CNN directly on a PyTorch implementation. These are weights have been made available online [4] by the authors of [3].

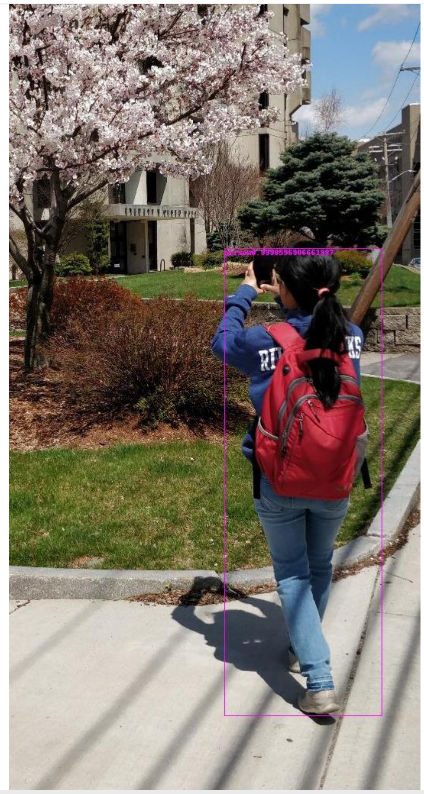
4.3 Performance evaluation

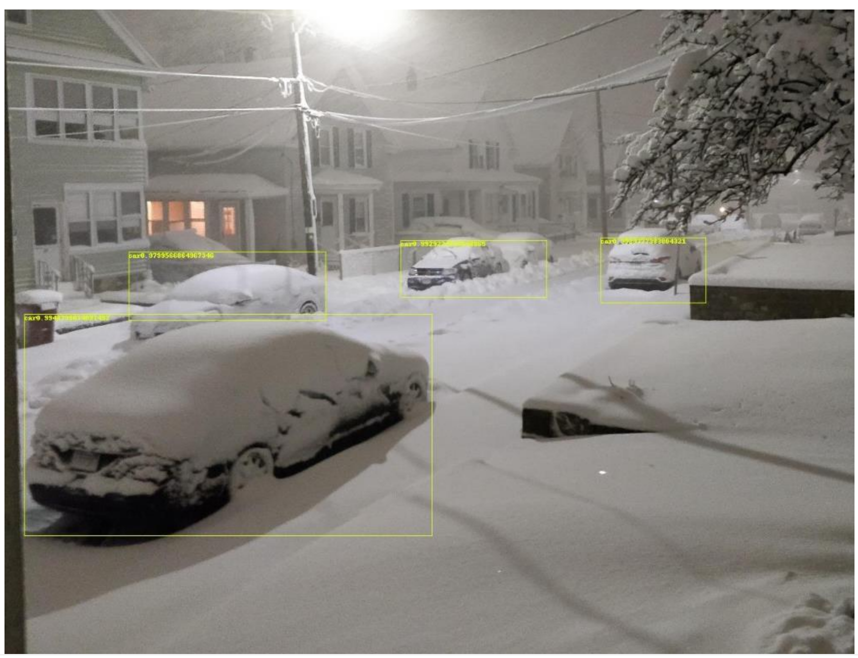
We tested our implementation using the PASCAL VOC 2007 test set and it achieved a **mAP score of 0.6422**. The authors of [3] achieved a mAP of 0.786. The evaluation/testing and score calculation was done by modifying the open source code shared by Github user Mavis[5]. We also thank Mavis for his description of darknet on PyTorch[6]. Fig 5a, b, c, d shows the output of images which are not from the VOC PASCAL dataset.

---

[1] http://pytorch.org
[2] http://host.robots.ox.ac.uk/pascal/VOC/
[3] https://github.com/pjreddie/darknet/tree/master/python
[4] http://pjreddie.com/media/files/yolo.weights
[5] https://github.com/marvis/pytorch-yolo2/blob/master/scripts/voc_eval.py
[6] https://github.com/marvis/pytorch-yolo2/blob/master/README.md







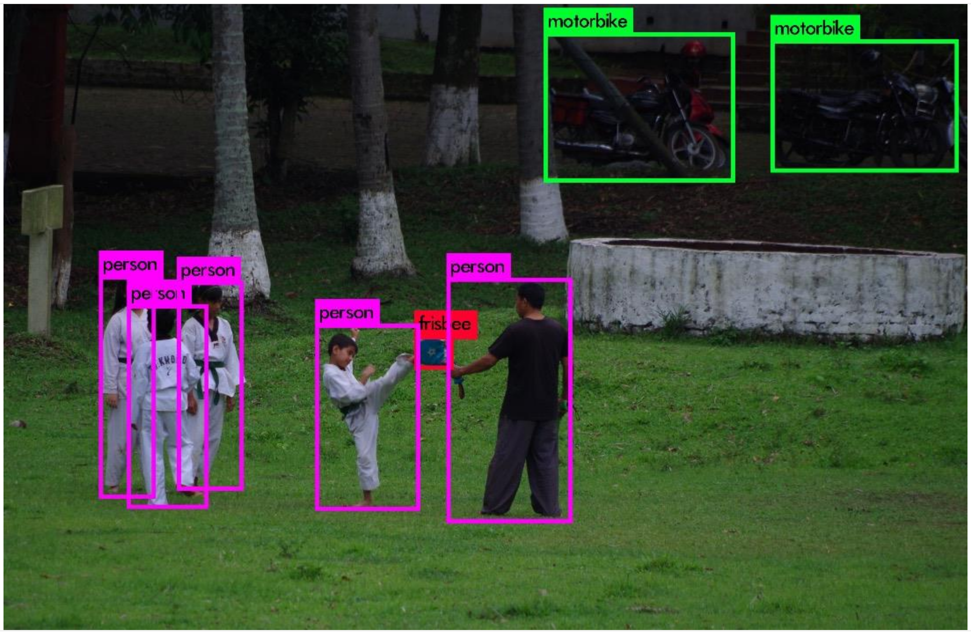


Fig 5 a, b, c, d Objects detected in images not from standard dataset.

We trained the system on a VM spawned in the Google Cloud platform. The VM had two 2.3 GHz Intel Xeon E5 v3 (Haswell) CPUs, 24 GM RAM and one NVIDIA Tesla K80 GPU which has 4992 CUDA cores (560 MHz) and two 12 GB RAM modules. Using the pretrained weights, we are able to run our detection program for single images on CPUs.

5. Conclusion

We have implemented a version of YOLO9000, an object detection system, in Python. It also give an RDF file as output. The system performs relatively well on thePASCAL VOC dataset. We were also able to get objects detected in images which are not from the dataset. In our proposal we had mentioned that we would generate an ontology for scene recognition and relative object relation. Unfortunately, we were not able to implement this. We were also not able to perform comparision of our implementation and other platform.

References

[1] Voila, Jones, 'Robust Real-time Object Detection', (2000). Second International Workshop on Statistical and Computational Theories of vision – Modelling, Learning, Computing and Sampling.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12), F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 1. Curran Associates Inc., USA, 1097-1105.

[3] Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 6517-6525.

[4] Redmon, J., Divvala, S.K., Girshick, R.B., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779-788.

[5] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus and Yann Lecun, (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell and Jitendra Malik, (2014). 'Rich feature hierarchies for accurate object detection and semantic segmentation'.

[7] Ross Girshick. 2015. Fast R-CNN. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV) (ICCV '15). IEEE Computer Society, Washington, DC, USA, 1440-1448. DOI=http://dx.doi.org/10.1109/ICCV.2015.169

[8] Redmon. Darknet: Open Source Neural networks in C. http://pjreddie.com/darknet/, 2013-

2016