Building Spatial Models

Khoshrav Doctor
University of Massachusetts
Amherst, MA
kdoctor@cs.umass.edu

Swapnil Debarshi
University of Massachusetts
Amherst, MA
sdebarshi@umass.edu

Abstract

This project attempts to build a model for spatial locations of objects in over time. This model can then be applied to solve problems in different domains and can act as search distributions, target distributions for the environment, priors for robot localization and also to learn inter-object relationships for planning and computer vision tasks. We overlay a real-time object detection model with a three degree of freedom robotic head to build this distribution in it's cumulative field of view. We evaluate this distribution by using it as a prior to search for objects in the environment.

1. Introduction

It is extremely trivial for human beings to subconsciously perform coarse object detection and localization while building an approximate spatial model of each object and it's distribution over both time and space. Humans then use that distribution as prior information when they are required to interact with these objects. These priors can allow robots to leverage known information to minimize computation and especially simplify multi-objective tasks. These models are represent the degree to which an object may lie in a discreteized sparse voxel grid.

Object detection is a type of image classification where each image can contain multiple object classes, and we also have to localize (draw bounding boxes) each object in that image. The initial object detection algorithms had two stage approach to object detection. The first stage would try to predict the object classes in the image using a general CNN network, while the second stage would try to predict bounding boxes around the image, using either a SVM [6] or a softmax layer at the end of the neural network [5]. Faster R-CNN used a separate region proposal network using fixed anchor boxes to propose different regions in the image which were then fed into an image recognition pipeline [10].

Object detection was still a static task because the above models were not suitable for real time object detection. This was made possible when object detection models could be built as single stage pipelines [8][9]. In our project we decide to use the YOLO Object Detector v2 for the object detection pipeline. YOLO [9] was primarily picked because it is able to match the detection accuracy of the Faster RCNN model [10] while processing images at the rate over 50fps. Also, unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.

2. Related Literature

Spatial models have been extensively studied in the context of object recognition. In [1], a spatial memory network is incorporated into Faster R-CNN to help object detection models make use of inter-object relationships. This helps the model take help of context reasoning in finding objects in a given scene. In [2], the authors propose a semi-supervised model that can be used to perform object recognition using training data that only provides information about class membership of parts by learning both a model of local part appearance and a model of the spatial relations between those parts. But as far as we know, there has been little research in building spatial models using object detection models that can be used for object querying or building inter object relationships.

3. Hardware

We use a 3 degree of freedom tilt-pan-tilt system with an Asus Xtion Pro RGB-D camera mounted the top. Each degree of freedom is controlled by a Dynamixel MX-28 Servo. Figure 1 shows the effect of each degree of freedom (top left- upper tilt, bottom right- pan, bottom left- lower tilt).

4. Implementation

For the actual implementation of the project, we first build our own implementation of the YOLO object detector using Python and Keras. We pick the YOLOv2 as the base

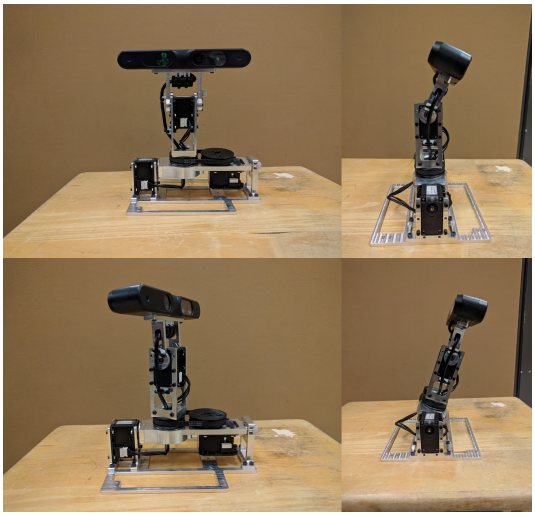

Figure 1: Hardware Setup

architecture of our model. YOLOv2 incorporates anchor boxes for drawing bounding boxes around objects as first suggested in the Faster R-CNN model. YOLO v1 used to divide each image into $S \times S$ grid, and used a 24 layer CNN followed by two fully connected layer. In v2, YOLO uses dynamic anchor boxes that are first found out by running k-means clustering on the bounding box information in the dataset to generate good priors for the model. YOLOv2 removes the fully connected layers and the entire architecture is just a 23 layer CNN that downsamples the 416×416 pixel image into 13×13 output tensor that contains information about all the bounding boxes as well as the classes probabilities for all the dataset labels. The entire model uses a custom loss function that penalizes incorrect labels around bounding boxes as well bounding boxes with no detected object classes.

4.1. YOLO Loss Function

The loss function is designed to take into account the dual application of the model to predict both the object classes as well as bounding boxes around detected object and help the network learn parameters to improve both these objectives concurrently.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i,j}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

$$+\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i,j}^{\text{obj}} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i,j}^{\text{obj}} (C_i - \hat{C}_i)^2$$

$$+\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i,j}^{\text{noobj}} (C_i - \hat{C}_i)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{I}_{i}^{\text{obj}} \sum_{\text{c } \epsilon \text{ classes}} (p_i(c) - \hat{p}_i(c))^2$$

where $\mathbb{I}_i^{\text{obj}}$ denotes if object appears at anchor point i and $\mathbb{I}_{i,j}^{\text{obj}}$ denotes that the j^{th} bounding box predictor at anchor point i is "responsible" for that prediction.

The loss function only penalizes the total error if there is an object in a box or the bounding box is actually responsible for the predictors (the ground truth box). We also penalize bounding boxes whose intersection over union (IOU) with ground truth boxes is less than 0.6.

4.2. Training

The weights of the first 20 Conv layers are pretrained on the ImageNet dataset [3], where it achieves a single crop top-5 accuracy of 88%, then it is trained on the COCO dataset [7] for Object Detection. For v2, the authors have decided to use ImageNet to make a hierarchical object model for all the 80 classes of the COCO dataset model to even try to detect granular objects in images. Since our project is not concerned with predictions at such a granular level, we retrain our own model using just the 80 classes of the COCO dataset.

To train the model, we download the pretrained weights from the YOLO website and retrain the last Conv layer for the 80 classes of the COCO dataset. We could not perform the full training due to resource constraints. We start with 100 epochs, but define an early stopping criteria that terminates training when 3 consecutive epochs does not result in validation losses that are better than 0.001. The weights of the best performing model for these 3 epochs are then used for the next stage.

On AWS p2.xlarge instance, our model trained for 11 epochs in 18 hours before reaching the early stopping criterion.

4.3. Integration with the Robotic Head

We use the Robotic Operating System (ROS Indigo) to execute multiple decoupled nodes. This system relies on

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			

Figure 2: Model Architecture

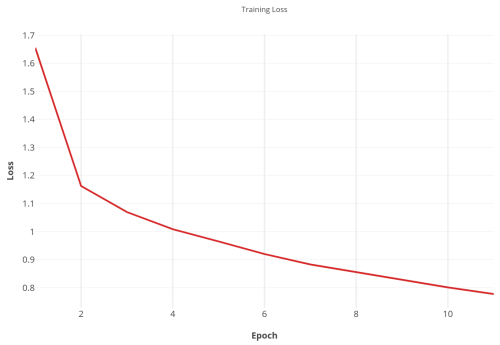

Figure 3: Training Loss

a set of nodes that perform their own computation, but may communicate with each other by publishing and subscribing to different topics. We use the following nodes to accomplish this task:

1. darknet_ros [4]: The ROS node provides a wrapper for darknet to use tiny_yolo for 2 dimensional RGB images. This package was altered to use the required models and publish the $(x, y, z)_w$ co-ordinates of the center of detected 2 dimensional bounding box. This node uses the RGB image obtained from the camera and localizes objects at 5 FPS. The center of each bounding box is found in the image plane and

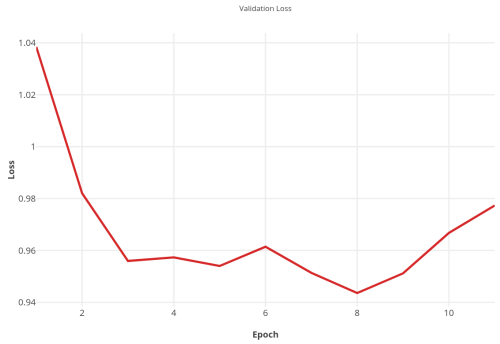

Figure 4: Validation Loss

it's corresponding $(x, y, z)_c$ position in the frame the point cloud was recorded in (referred to as the camera frame). Using the known parameters of the robot (link lengths and joint angles) we then use a sequence of homogeneous transforms to transform this point into the world frame using the process of forward kinematics. For each detected object, a message is published that records the position in the world frame, the class label and the probability outputted by the network. To improve the performance, only instances that have a probability greater than 0.5 are published.

2. head_server: This node is written to interface with the hardware. This node receives desired joint angles from other nodes and sets them to the hardware. Additionally, it is also responsible for reading the current joint angles off the hardware (which is required to generate the transforms between different frames of reference).

3. gaze_controller: This node is responsible for generating and publishing desired joint angles required to make the camera look at a point in space given by $(x, y, z)_w$. It does so by projecting a unit vector along it's current gaze direction and the desired gaze direction and then descending on a potential field that minimizes the angle between them. As a secondary controller it also picks joint angles that attempts to minimize the deivation of each joint from the center of its range of motion.

4. head_search: This node is what is actually responsible for building and using the model. Unlike the other nodes that run at extremely high frequencies, this node is made to run at a 0.1 cycles per second. This is done to avoid noisy reading during the movement of the head and allow for a stable image to build the model. For each entity (class label) there are two voxel maps created. Each voxel is a 0.1m $\times$ 0.1m $\times$ 0.1m bin that stores the likelihood of the entity being present in that voxel. The first maps store the unnormalized count of

the entity's center being present in the bin. The second map takes into account potential errors in YOLOs prediction. To do so we compute a moving average of the propbabilities within that voxel. More formally, for an entity *e*, at point *(x, y, z)* detected with probability *p*:

$$prob(e,x,y,z)_{t+1} = \\ ((prob(e,x,y,z)_t * count(e,x,y,z)) + p)/ \\ (count(e,x,y,z)+1) \quad (1)$$

$$count(e,x,y,z) = count(e,x,y,z) + 1 \quad (2)$$

There are two main modes of operation (to build the model or to search using it).

(a) Model building phase: Here, for every cycle, we generate a random point in space $(x,y,z)_{desired}$ and use the gaze_controller to compute and publish the corresponding joint angles to the head_server. The random point lies anywhere in front of the head (i.e.$-\pi/2 \leq$ pan_angle $\leq \pi/2$). Messages from darknet_ros are processed and used to update the 2 voxel grids, and the cycle is then repeated.

(b) Seach: During the Search phase, the user has the ability to specify the label they wish to search for along with an indicator of which map to use. The appropriate grid is then sorted in descending order of likelihood. This list is then used to provide goals to the gaze_controller. The search is terminated if the object is found using messages from darknet_ros or if the list is exhausted. The condition for termination, number of actions taken, and the euclidean distance between the voxel being looked at and the actual position of the found entity is reported to the user.

5. Experimental Results

To perform the experiment, the scene in Figure5 was used. The potentially observable classes were backpack, book, bottle, chair, cup, diningtable, laptop, person, sofa and tvmonitor. The model was built over a 5 minute interval. The environment was dynamic and the position of the cup and bottle was changed during this course. Additionally, some entities (chairs, tvmonitors) had multiple scattered through the environment. Using this model, a search was performed using each map and a uniform prior as a baseline. One action is defined as the movement to one point and the corresponding scan of the image. A search was terminated if the object was not found within 25 such actions. Table 1 shows the results for a few searches.

	Entity	Count	Likelihood	Baseline
1	Backpack	1	1	-
2	Bottle	2	1	4
3	Chair*	12	6	5
4	Sofa	1	1	1

Table 1: Search Actions required to find entity

For objects that were stationary and approximately present in the center of the field of view, all 3 methods proved to be equally efficient. For the moving objects, the probability based metric appears to work better. This can be attributed to the fact that false positives are suppressed and not overpowered by multiple incorrect observations.
It should be noted that the backpack was not found by the baseline method in 25 actions. The experiment to search for the Chair was purposely made adversarial to highlight situations where this form of modelling may not be as effective. While building the model, chairs were present in the center of the field of view and at the extremes. Before the search, all chairs were removed from the scene except for the one that was observed the least number of times. In this scenario, it would be expected that the models would have to search through all the positions until the last while the uniform prior might allow for it to be found sooner.

Figure 5: Experimental Setup

6. Future Work

The implementation of this project has brought to light potential improvements. The most important begin an intelligent way to normalize the count map. This map does not account for situations where there is a low count simply because that voxel has not been looked at frequently. Although this project aims only at *building* these distributions and not using them, the results might have proven to be more effective if a more intelligent search strategy was used. If the model could be diffused when objects were **not** found in voxels surrounding a region, it could eliminate actions that require it to look at points that have already been in the field of view (but not foveated on). The model is also sensitive to the discretization of the voxel grid. This is apparent in the experiment to search with the chair since a large portion of the actions were to look at voxels neigh-

bouring other searched locations.

7. Supplementary Material

Code and supplementary material can be found at `https://github.com/khoshrav/spatial_model_builder`

References

[1] X. Chen and A. Gupta. Spatial memory for context reasoning in object detection. *CoRR*, abs/1704.04224, 2017.

[2] D. J. Crandall and D. P. Huttenlocher. Weakly supervised learning of part-based spatial models for visual object recognition. In A. Leonardis, H. Bischof, and A. Pinz, editors, *Computer Vision – ECCV 2006*, pages 16–29, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[4] P. Gigioli. darknet_ros. `https://github.com/pgigioli/darknet_ros`, 2017.

[5] R. Girshick. Fast R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2015.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[7] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing.

[9] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[10] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems (NIPS)*, 2015.