EECS473 Project Report

dxg397

December 2018

1 Introduction

Detecting classes of objects reliably remains a highly challenging problem in robotics. Robotic systems require extremely high accuracy for a large variety of object classes in order to perform even simple tasks (such as taking inventory of a few objects in a lab environment) but off the-shelf vision algorithms that achieve the necessary level of performance do not yet exist. Robots, however, have a number of advantages that are ignored by most detection systems. For instance, robots are able to view a real scene from multiple angles and can view interesting objects up close when necessary, yet most existing algorithms cannot directly leverage these tools. In this work, I have presented a method which uses Fast Region-based Convolutional Networks for object detection and ROS (Robotics Operating System) to serve as a service for the Object detection system.

Fast R-CNN is a fast framework for object detection with deep ConvNets. Fast R-CNN

- trains state-of-the-art models, like VGG16, 9x faster than traditional R-CNN and 3x faster than SPPnet,
- runs 200x faster than R-CNN and 10x faster than SPPnet at test-time,
- has a significantly higher mAP on PASCAL VOC than both R-CNN and SPPnet,
- and is written in Python and C++/Caffe.

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

2 Setup

This section describes the software setup that was used during development. Regardless of that, adapting the code to other robots, cameras, or ROS versions should require only minor changes. The object recognition software can also be used without any robot.

The notebook of the Bot was running Ubuntu 16.04 and ROS Kinetic.

The libraries that were mainly used are OpenCV 3, PCL 1.7, Qt 5, and Eigen, Caffee, py-faster-CNN. Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors. Yangqing Jia created the project during his PhD at UC Berkeley. Caffe is released under the BSD 2-Clause license.

The system employs both py-faster-rcnn and py-R-FCN as a ROS package for object detection. It can detect the at least 20 object categories. Moreover, it contains a fine-tuned model for detecting chairs, laptops and monitors. I have used few pretarined Caffee models available online to detect the objects.

- VGG16-faster-rcnn-final.caffemodel
- ZF-faster-rcnn-final.caffemodel
- Resnet101-rfcn-final.caffemodel
- Resnet50-rfcn-final.caffemodel
- Resnet101-rfcn-coco.caffemodel

Few of the Objects it correctly classifies are : 'aeroplane', 'bicycle', 'bird', 'boat','bottle', 'bus', 'car', 'cat', 'chair', 'keyboard', 'diningtable', 'dog', 'horse', 'motorbike', 'person', 'bottle', 'cup', 'sofa','train', 'tv', 'monitor'.

3 Message type for Detected Objects

The importance of uniform, well-defined interfaces for specific kinds of modules has already been pointed out. Obviously, detection modules have to use images, point clouds, or similar data as their input. They also have to publish the detected objects in some way. This section describes the message types that were defined for this purpose.

The message type that is used for detected objects has to be sufficiently general to be useable by a large number of detection modules. The type should allow to describe the pose and size of a detected object accurately enough for most applications. On the other hand, the description should be simple enough to be easy to use. To keep the network load low, the reprentation of objects also should not require an unnecessary amount of data.

- Detected object Meassage
  - deep-object-detection/Object[] objects Detected objects
  - string observation-path The observation path
- Object message
  - Header header
  - string label # label of the object
  - uint32 x # x coordinate of the anchor position of bounding box
  - uint32 y # y coordinate of the anchor position of bounding box
  - uint32 width # width of the bounding box
  - uint32 height # height of the bounding box
  - uint32 imageID # the id of the image that the object belongs to
  - float32 confidence # the confidence value of the neural net

4 Heart of the System

Here is the heart of the code. I tried to comment it pretty well, but here is the workflow.

- The callback function fires when a new image is available.
- Use cv-bridge to convert the image from a ROS image type to an OpenCV image type.
- Resize the image to the shape required by ResNet50, 224 x 224.
- Read the OpenCV image in as a NumPy array.
- Expand the array into the size needed for TensorFlow.
- Convert the data from uint8 to float64.
- Normalize the data.
- Run the model and classify the image.
- Decode the prediction and convert them to appropriate data types.
- Publish the prediction.

5 Usage

- Use the download-models.sh script, to download the Caffe model files.
- You can run the node using the command: rosrun deep-object-detection object-detection.py

By default the node will run in CPU mode. In order to run on the GPU, you should set the gpu flag as –gpu GPU-ID where GPU-ID is the id of the GPU that you want to use which is usually 0.

- By default VGG16 network will be used but you can also use the ZF network using –net zf or the fine-tuned 3-class KTH model using –net vgg16KTH.
- For using the RFCN models, you should use –net ResNet-50 or –net ResNet-101 commands.

6 Services

There are 2 services advertised by this node:

- /deep-object-detection/get-labels This service will return you the list of object labels that can be recognized. There are 20 labels.
- /deep-object-detection/detect-objects This service will return the detected objects as a vector of images. The bounding box, label and confidence information of each detected object is returned.

7 How Caffee model works

In our experiment, we used the VGG 16 as the base network. An auxiliary structure was appended to the network to produce detections with the following key features:

- Multiscale feature maps for detection: We added convolutional feature layers to the end of the truncated base network. These layers decreased the size progressively and allowed predictions of detections at multiple scales.
- Convolutional predictors: Each added feature layer can produce a fixed set of detection predictions using a set of convolutional filters. For a feature layer of size mn with p channels, the basic element for predicting parameters of a potential detection is a small 33p kernel that produces either a score for a category, or a shape offset relative to the default box coordinates. At each of the mn locations where the kernel is applied, it produces an output value. The bounding box offset output values are measured relative to a default box position relative to each feature map location.

- Default boxes and aspect ratios: We associated a set of default bounding boxes with each feature map cell for multiple feature maps at the top of the network. The default boxes tile the feature map in a convolutional manner, so that the position of each box relative to its corresponding cell is fixed. At each feature map cell, we predict the offsets relative to the default box shapes in the cell, as well as the per-class scores that indicate the presence of a class instance in each of those boxes. Specifically, for each box out of k at a given location, we computed c class scores and the four offsets relative to the original default box shape. This resulted in a total of (c+4)k filters that were applied around each location in the feature map, yielding (c+4)kmn outputs for an mn feature map.

The video captured by the camera was broken down into frames using OpenCV with a configurable frames per second. As the frames were generated, they were passed to the detection model, which localized the different objects in the form of four coordinates (xmin, xmax, ymin, and ymax) and provided a classification score to the different possible objects. By applying the NMS (Non-Maximal Suppression) threshold and setting confidence thresholds, the number of predictions can be reduced and kept to the prediction that is the most optimal. OpenCV was used to draw a rectangular box with various colors around the detected objects

8 Conclusion

The functional use case attempted ,involved the detection of objects and person from a camera or webcam. The training data was more skewed towards daily life objects as opposed to other objects of interest since it was from the pretarined models. The use case could be further expanded for video surveillances and tracking, finding the right object to pick using a robot, etc.[1]

References

[1] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.