Neural Network-Based Object Detection System for RoboCup

Jack Beckitt-Marshall, Class of 2021

In the summer of 2018, I worked with Professor Eric Chown to develop a new object detection system for RoboCup, using neural networks to process images taken by the cameras used by the Aldebaran Nao robots and find bounding boxes of the objects detected within the images.

A former student at the College, Konstantine Mushegian '17, suggested that the YOLO (standing for You Only Look Once) neural network architecture was a good fit for our task. The initial approach I was going to use was to use a classifier, a type of neural network that takes an image and returns probabilities of what the object within the image is, and simply make a "box" that slides across the image and scales itself, finding the locations of the objects with the highest probabilities. However, this approach is slow, as we have to evaluate the image using a neural network hundreds or even thousands of times. YOLO, however, takes a different approach, where the neural network is applied to the entire image, dividing it into regions and predicting bounding boxes and their probabilities for each region, with bounding boxes being weighted by their predicted probabilities (Redmon and Farhadi 2018).

What made YOLO attractive for this project was the speed: compared to other models that can only process 10-20 images per second, YOLO is around 2-4 times as fast. There is, however, a version of YOLO called Tiny YOLO, trading off accuracy for speed: the mean average precision is lower by approximately half, but the speed is much greater – around 5-6x as fast as standard YOLO (Redmon and Farhadi 2018). While these speeds definitely look promising, the caveat is that these speeds were achieved a high-end graphics processing unit. The Aldebaran Naos, meanwhile, do not even have graphics processing units, meaning that we must use the central processing unit (CPU), which is much slower than even the most basic GPU for neural network processing. Therefore, the biggest challenge with this project would be how fast in practice the neural networks would be when running on the robot itself.

The first step in the implementation of developing the object detection system was to gather images, which would then be used to "train" the neural network to detect objects. I decided that there would be five classes of objects that would be useful for the robots to detect: other robots, the heads of other robots, soccer balls, goalposts, and the center circle on the field. To gather enough training data, I used the internet, using image databases such as ImageNet, as well as taking log images using the robots themselves and using a camera. However, the dataset I would be able to gather would be relatively small, so I decided to create a project, using Python and OpenCV, that would essentially place the objects on different backgrounds, in random positions, and perform transformations on each image (such as blurring and pixellating). This allowed me to expand my dataset from a few hundred images to more than 10,000 images

Initially, I was going to use an implementation of YOLO using the TensorFlow library, developed by Google, which offers many advantages: it is well-documented, can use both CPUs and GPUs for neural network operations, and most importantly, integrates well with technologies already used by the RoboCup team. However, TensorFlow relies on a newer standard of the C++ programming language that the Nao robots do not support. Therefore, while I trained the network initially with TensorFlow, I eventually made the decision to use the Darknet neural network library, written by the authors of the YOLO paper, which is less well-documented, but was able to run on the robots.

Training the neural network was much faster than anticipated, as I took advantage of Bowdoin's HPC grid. When training using Darknet, a number of training "checkpoints" were produced, and I selected a checkpoint with a low loss, but not the minimum loss to use as the final weights, as the checkpoint with minimum loss had the potential to be "overfitted" – where the trained model is so tailored to the original dataset that it struggles to make new predictions.

The next, and perhaps final step of the project was to implement the object detection system on the robots themselves. This meant analyzing the code that we already had as a team and adding code that would pass the image to the neural network, which would then analyze the image and return bounding boxes. I managed to do this using the C++ language. However, when running the neural network on the robot, it takes approximately 14 seconds to process a single image, giving us an effective processing speed of around 0.07 frames per second, which is too slow to be useful in any capacity. One solution would be to offload the processing onto a faster computer, but this is disallowed under RoboCup SPL rules.

Work I could do to expand this project further includes using different neural network models such as Google's MobileNet, which may be better optimized for the embedded system inside the robot, or try and optimize the code even further, including potential use of routines coded in assembly language and optimized specifically for the processor in the Nao robots.

Faculty Mentor: Eric Chown
Funded by the Kibbe Science Fellowship

References:

Redmon, Joseph, and Ali Farhadi. 2018. *YOLOv3: An Incremental Improvement*. Academic paper, Seattle: University of Washington.