CS-8803 Special Topic: Machine Learning for Robotics

Second Intermediary Project: Using Convolutional Neural Networks

Support: cedric.pradalier@georgiatech-metz.fr
Office hours: 10:00 to 18:00.

Setup

This project use the 3D scene:

```
/cs-share/pradalier/scenes/lakeExample.ttt
```

The objective of this project is to give you an experience of setting up a convolutional neural network using a standard library called *Caffe*. In particular, you will be using the following example page from Caffe:

http://caffe.berkeleyvision.org/gathered/examples/imagenet.html

Because setting up Caffe and integrating it into a personal project requires a significant experience, a lot of this has already been done for you into the packages `floor_plane_deep` and `shore_follower`.

Also, deep-learning requires access to machines with large GPUs. We have 3 such machines at GTL in the server room (192.93.8.191/192/193), all equipped with a Nvidia K20 (approx. 2500 cores and 5GB of RAM). To work on these machines, you will have to use SSH and I strongly recommend using byobu, rsync and git. Everything required is installed on these machines.

Project Objective 1

The first objective of this project is to build an appearance-based obstacle detection system. Such a system cannot be robust for real-world use in general, but it provides a convenient problem to test our deep-learning framework. Use `floor_plane_deep` as a starting point.

Any learning-based solution must be implemented in two stages: first we need to build a training dataset (and split it between training and validation data) and then we can use the trained network to do runtime classification.

Building the dataset

The launch file fp_acquire.launch starts the binary generated from floor_plane_extract.cpp. This bit of code is already designed to acquire synchronously images and point cloud from the kinect floating on top of the bubbleRob. For each images, it then create a set of 32x32 pixel thumbnails with image data and height of the pixels over the ground from the kinect data (frame conversion is already implemented). You need to modify the check_thumb function, to use the kinect data to decide of a label for the thumbnail. The label can be “traversable”, “untraversable” or “unusable” if the thumbnail does not have enough 3D point in it (pixels for which no height have been observed have a value of NAN).

Once check_thumb returns, the usable thumbnails are saved as PNG files and a label file is updated. Check the out_dir parameter in the launch file.

When this process is ready, you need to drive the bubbleRob around the environment to provide a sufficiently diverse training set.

You then need to log on your accound on the GPU machine, convert the data into the Caffe format, compute the image mean and train your network. You may have to adjust the scripts a little to account for the specifics of your paths. Check the imagenet link above for the process that need to be followed. Note that the network structure has already been provided in the models directory. It is a small network, suitable for the small images we are considering. Because it is such a small network and such a big GPU, training will be over in only 10-15 minutes, maybe less if you tune the parameters in solver.prototxt (in the models directory). Please note that the provided model (train_val.prototxt) has a small bug that you need to correct before training the network.

Using the network

Once training is complete, you can download the caffemodel file back to your workstation and using the launch file fp_classify to test the performance. Before that however, you need to correct another small bug in the model specification (deploy.prototxt) and modify the check_thumb function in the floor_plane_classifier.cpp function to convert the output of the network (defined in deploy.prototxt in the models directory) into our traversable or untraversable labels. Once the function returns these labels are used to color the image based on the predicted danger. Use the standard ros tool image_view (or rviz) to subscribe to the published image.

Deliverables:

- Modified source code
- Presentation and demonstration highlighting the strategy to build the dataset and the performance evaluation criterion.
- Quantitative and Qualitative performance analysis

Project Objective 2

The second objective of this project is to train a second network to drive the bubbleRob along the shore of the dried-up lake. To this end we will train a discrete network that can output 3 labels: go straight, turn left, turn right. The base package is provided in the `shore_follower` package.

Building the dataset

The first launch file fp_acquire.launch starts the binary generated from the shore_follower_observe.cpp code. This code subscribe to the image published by the kinect and to the command sent to the bubbleRob from the joystick. You can enable/disable the recording by pressing the orange button. Your goal is to demonstrate to the bubbleRob how to drive around the lake. When the recording is active, the program will simply save every image it receives, associated with the discretized command you applied. The threshold for this discretization is one of the parameter you need to tune but there is no need to modify the source code. max_image_per_type will control the number of images recorded for each label. For a good learning, you need a balanced number of example and you need to show as many situations as possible, in a balanced way as well.

Once the dataset is acquired, you need to log onto the GPU machine and train the network following the same process as above. The scripts and models with a _fast suffix will resize the image to a 32x32

pixel thumbnail. This is sufficient for the control and let the network learn in a shorter time. Note that the train_val_fast.prototxt contains some little bugs that need corrected before starting the learning process.

Using the network

When the learning is complete, you can once again bring the model back and test it. Before that, you need to modify the image_to_rot function in the shore_follower_drive.cpp code, to interprete the network output.

At this stage, you will probably find out that your dataset may not have been complete enough or you labels and threshold not set correctly. Keep the model aside for comparison and iterate.

Deliverables

- Modified source codes
- Presentation and demonstration of performance highlighting the strategy to build the dataset and the performance evaluation criterion.
- Quantitative and Qualitative performance analysis and in particular, comparison of shore following performance between different network trained in different conditions (distance to the shore, thresholds, island vs outer rim, …).