Crowd Density Manual

Author: Anthony G. Musco

Email: amusco@cs.stonybrook.edu

Updated: May 15, 2017

Introduction

There are quite a few interacting components for this project which run in different environments and on different machines. To start, we should clarify that the different operations for this project are designed to run on two different machines: `local` and `server`. When we refer to `local`, we mean on your local machine, in the project directory. When we refer to `server`. we mean SSH'd into `bigbrain` in the Vision lab. The project includes several scripts to automate certain operations, and all of them should be executated on your `local` machine. If needed, the script will open up a SSH connection to the server to execute desired tasks.

Setup

Local

Project Directory

Clone the project directory from the remote repository to your desired local directory. The project uses Python 2.7, so create a virtual environment if your native Python is different. You can check via the following:

```
user@local:~$ python --version
2.7.12
```

Docker

Install Docker on your machine, then pull the image for `Gym-Gazebo`:

```
docker pull erlerobotics/gym-gazebo:latest
```

Be patient, this process might take a while. You can enter the image by running in interactive mode (enter CTRL-D to exit when you are done):

```
docker run -it erlerobotics/gym-gazebo
```

Gazebo

Gazebo is required to generate the images on the local machine. You should install Gazebo on your machine following the instructions at the following address: Install Gazebo.

Server

Before you begin, you need SSH access to the `bigbrain` server, preferably set up with authentication keys. Many of the scripts assume a `bigbrain` host in your `~/.ssh/config` file, so add the following to `~/.ssh/config` if you have not already done so (Note: in all of the below commands, you should replace `user` with your assigned user name):

```
Host bigbrain
  HostName bigbrain.cs.stonybrook.edu
  Port 130
  User user
  ForwardX11Trusted yes
```

Make sure this works by executing the following and making sure you can properly access the server (you can enter CTRL-D to exit the server when you are done):

```
# On your local machine.
user@local:~$  ssh bigbrain
# Now on the server.
user@bigbrain:~$
```

Next you should create a directory on the NFS for the project directory, and make a symbolic link to it from your home directory:

```
user@bigbrain:~$ mkdir /nfs/bigbrain/user/crowd_density
user@bigbrain:~$ ln -s /nfs/bigbrain/user/crowd_density ~/crowd_density
```

TensorFlow

This project uses TensorFlow 1.0.1, which is not supported on the server. To use this version, we create a Python virtual environment using the following commands:

```
# Copy Python requirements file to server.
user@local:~$ scp crowd_density/tensorflow-requirements.txt
user@bigbrain:~/crowd_density
# SSH to server.
user@local:~$ ssh bigbrain
# Create virtual environment and activate it.
user@bigbrain:~$ virtualenv ~/tensorflow
```

```
user@bigbrain:~$ source ~/tensorflow/bin/activate
# Install required packages.
(tensorflow)user@bigbrain:$ pip install -r
~/crowd_density/tensorflow-requirements.txt
```

As of this writing the server's verion of CUDA is 8.0, while TensorFlow is compiled to use CUDA 7.5. To remedy this, we need to update the `LD_LIBRARY_PATH` variable as follows (Note: all the server scripts in the `scripts` directory do this automatically, so you shouldn't need to do it yourself):

```
# Update variable before activating virtual env.
user@bigbrain: LD_LIBRARY_PATH=/usr/local/cuda-7.5/lib64:$LD_LIBRARY_PATH
# Activate virtual env before running TensorFlow.
user@bigbrain:~$ source ~/tensorflow/bin/activate
(tensorflow)user@bigbrain:$
```

Now, everything should work. To test, try importing TensorFlow in Python and verifying the version:

```
(tensorflow)user@bigbrain:$ python
>>> import tensorflow as tf
>>> print tf.__version__
1.0.1
```

Project Structure

The main project consists of the following directories:

- `ccnn`: Python module for the Counting CNN, implemented in TensorFlow.
- `config`: Cross-project configuration properties.
- `data`: Raw images and example files, as well as set lists for each.
- `docs`: Documentation and papers.
- `gazebo`: Gazebo plugin code for data generation.
- `gym`: OpenAI Gym code.
- `logs`: Logs for training/testing the Counting CNN. This directory also contains the trained model at `logs/train/checkpoint`
- `scripts`: Useful scripts for performing common tasks.

Scripts

All scripts should be run from the root project directory. In the descriptions below, we omit the leading directory for brevity. For example, the script `generate-images.sh` should be run as follows:

```
# Generate images.
```

```
user@local:~/crowd_density$ scripts/generate_images.sh
```

Generating Images

The generation of images takes place on the local machine, as it requires Gazebo. The number of images generated is controlled by the `IMG_NUM` parameter in the configuration file.

If Gazebo is properly installed, you should be able to generate images using the `generate-images.sh` script from the root directory. The script first builds the gazebo project using CMake if necessary, then sets the environment variables, then runs the Gazebo world file with the custom plugins. If done correctly, the new images and image set lists should be output to the `data/images` and `data/image_sets` directories, respectively. To clear these directories and generate new images, use the `clear-images.sh` script (Note: this operation deletes the images and cannot be undone.)

Pushing to Server

Onces images have been generated, they can be transfered to the server using the `server-push.sh` script. The push operation performs a synchronization between directories, so only diffs are sent on the network. It may take some time to push everything the first time, but after that pushing will be relatively quick. You must push to the server any time you alter source code, configuration params, or data.

Generating Examples

Examples are binary files containing a (patch, density) image pair that can be easily read in a queue by TensorFlow. We generate them ahead of time to make training go by faster.

Examples are generated on the server to avoid storing GB's on the local machine. You can run the `server-generate-examples.sh` from the local machine, which SSH's to the server and runs the `ccnn/generate-examples.py` script to generate examples from the images.

Training

Once the examples have been generated on the server, you can begin the training session by running the `server-train.sh` script from the local machine. This will automatically SSH to the server, activate the tensorflow environment, and run the `ccnn/train.py` Python script to commence training. Training outputs logs and checkpoints to the `logs/train` directory on the server, and is fault tolerant. If the training session gets interrupted for any reason - be it a bug or lost connection, simply re-run the script and the session will pick up right where it left off. To start from scratch, delete the contents of the `logs/train` directory (Warning: Doing so will delete any previously trained model you have).

TensorBoard

TensorBoard is an immensely useful tool for monitoring a training session, and can be run on the

server to monitor progress. After you start a training session, open a new terminal window and execute the `server-tensorboard.sh` script from the command line. If all goes well, you should be able to view the running training session from your browser, at `http://127.0.0.1:6006`.

Note: when the `server-tensorboard.sh` script exits (CTRL-C), there is a possibility it may leave a lingering TensorBoard process on the server attached to port 6006. To check, you can inspect the port using `fuser`:

```
user@bigbrain:~$ fuser 6006/tcp
```

If the line is non-empty, simply call `kill -9` using the process ID it returns.

Pulling from Server

The opposite of the `server-push.sh` script is the `server-pull.sh` script, which pulls log data from the server to your local machine. Note that the `train/log/checkpoint` file is your trained model, so it is needed if you would like to use the trained Counting CNN on your local machine (or anywhere, for that matter).

Testing

After training your model, you can see how it performs on hold-out test data by running `server-test.sh`. This will produce a histogram image on the server showing how your model performed across scales. The image will be pulled down with the `server-pull.sh` script.

Inspecting Model Performance

Once the `logs/train/checkpoint` file is pulled down, you can see how it performs by running the `inspect-data.py` Python script. This will overlay the ground truth and predicted depth maps, and compare them side by side. Take this output to note what the Counting CNN is learning, and whether it is generalizing to different scales.