Lab 4: Autonomous Driving

Learning Outcomes

- Acquire training data for Machine Learning
- Design a Neural Network with Caffe
- Train the Neural Network using a GPU
- Perform inference to navigate a course

Section 1: Requirements and Design

In this lab, you will train a Neural Network to navigate a course autonomously.

Section 2: Installing the Lab

To perform this lab, you will need to get the lab4_autonomous_driving template into your catkin workspace. First ensure that your Jet has internet access by connecting it using WiFi or ethernet. Next ssh into Jet and enter the following command:

```
wget http://instructor-url/lab4_autonomous_driving/lab4_autonomous_driving-code.zip
```

Where the url should be replaced by the URL provided by your instructor. Now unzip the lab:

```
unzip lab4_autonomous_driving-code.zip -d ~/catkin_ws/src/jetlabs/lab4_autonomous_driving
```

Delete the zip file:

```
rm lab4_autonomous_driving-code.zip
```

To build the code, use the following command when you are in `~/catkin_ws/`:

```
catkin_make --pkg lab4_autonomous_driving && source devel/setup.sh
```

To run the system in training mode, execute the following

```
roslaunch lab4_autonomous_driving lab4_train.launch
```

To run the system in inference mode, execute the following

```
roslaunch lab4_autonomous_driving lab4_inference.launch
```

Section 3: Acquiring Training Data

The first step in building a successful machine learning model is to acquire training data. The input to the Neural Network will be frames from the robot's camera. The output of the Neural Network will be the direction that the robot should go (FORWARD, LEFT, or RIGHT). We will generate training data by manually driving the robot through the course and recording each video frame along with the current direction of the robot.

First, you should practice driving the robot in the course. Launch the jet system:

```
roslaunch jet_bringup jet_real.launch
```

Then in a separate terminal, run the keyboard teleop node:

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Use the i (forward), j (left), and l (right) keys to move the robot. Once you are comfortable moving the robot, you can begin a training session. The `drive_train` node records the video frames and velocity as you navigate. You can run `drive_train` with the following command in a new terminal:

```
rosrun lab4_autonomous_driving drive_train
```

When you have successfully navigated the course several times, quite all of the open ROS nodes. Now you can convert the raw data into a format that will be used by the Neural Network with this command:

```
rosrun lab4_autonomous_driving preprocess.py
```

Section 4: Designing and Training the Neural Network

Now you must design a Neural Network that will learn to identify when the robot should go left, right, or forward. You can use your notes from the lecture on Caffe and the Caffe Model Zoo as inspiration. Remember that more complicated networks can take longer to train and are more prone to overfitting. Implement the layers of your Neural Network Design in `neuralnetwork/architecture.prototxt` and `neuralnetwork/deploy.protoxt`. The layers that you enter in both of those files must be identical. The final layer of your Neural Network must have three outputs (forward, left, and right).

The file `neuralnetwork/train.prototxt` is where you can specify how the Neural Network should be trained. You can modify these parameters to modify how the Network is trained; below is an overview of the important values:

- base_lr: learning rate that updates the weights (higher number means faster training, but if the number is too high, the Neural Network may not converge)
- test_iter: number of batches to evaluate when assessing test accuracy
- test_interval: how many epochs between accuracy evaluations
- max_iter: maximum number of epochs to train the Network
- momentum: the learning parameter that helps the Neural Network escape local optimums
- snapshot: the number of epochs between the snapshots of the Neural Network's weights

To train the Neural Network, run the following:

```
rosrun lab4_autonomous_driving train.py
```

If the loss value begins to increase, then you will likely want to decrease your learning rate. If the process hangs, then your Neural Network might be overly complex or your batch size is too large. You can decrease the batch size in `neuralnetwork/architecture.prototxt`. You can stop the Neural Network trainign at any point with CTRL-C.

Section 5: Testing the Neural Network

After the Neural Network has been trained, the model weights are stored in neuralnetwork/models/train_iter_<I>.caffemo where is the last epoch. Modify `src/drive_inference` to point to the correct model that you trained. Then run the inference launch file:

```
roslaunch lab4_autonomous_driving lab4_inference.launch
```

Place the robot in the course and open the direction topic in a new terminal:

```
rostopic echo lab4_autonomous_driving/dir
```

Ensure that the robot is correctly outputting the desired direction when the robot is point forward, left, and right. If the system is incorrectly identifying the direction, then you may need to collect more training data, train the network for longer, or modify your network design. Once you are satisfied with the accuracy of the Neural Network, you can proceed to the final step.

Section 6: Driving Autonomously

Within `src/drive_inference.cpp`, insert code in the imageCallback function that publishes a new velocity vector based on the Neural Network's output. Recompile the package and run the inference again:

```
roslaunch lab4_autonomous_driving lab4_inference
```

If your Neural Network and code work properly, the robot should be able to navigate the course without human assistance.

Section 7: Bonus Challenge

Try pointing your robot in the opposite direction to see if it can still navigate the course. If it cannot, then try adding training data in the reverse direction and retraining the network.