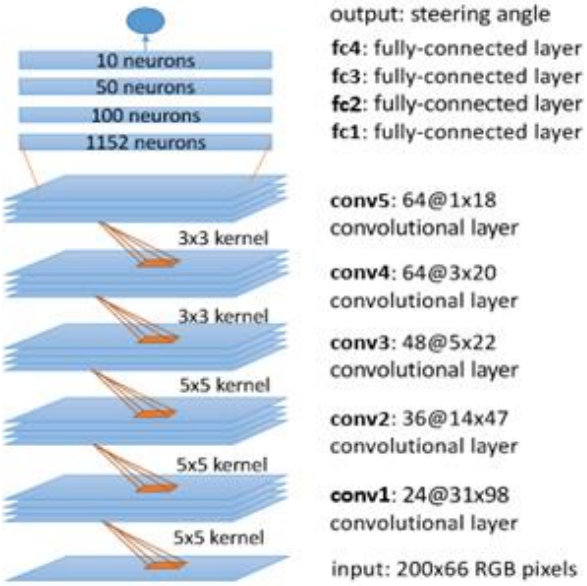
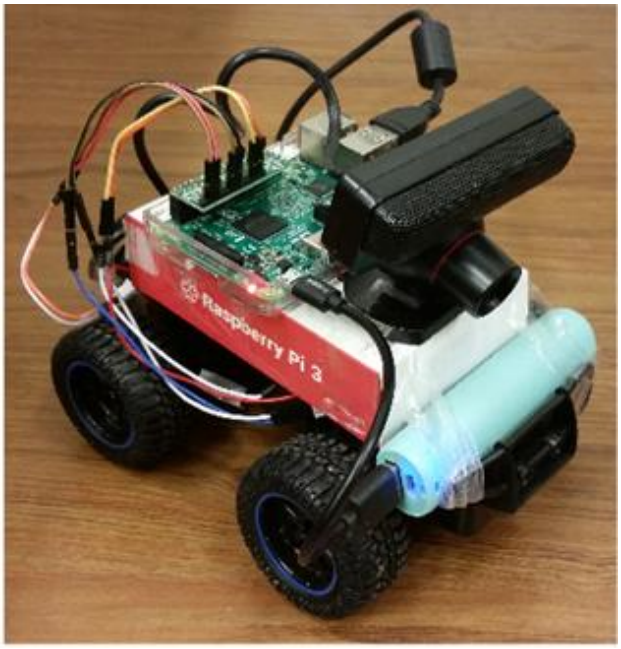
EECS 388 Lab #7

Real-Time DNN Inferencing

In this lab, you will learn how to load a Deep Neural Network (DNN) model and perform inferencing operations on the Raspberry Pi 4.





(Note that much of this lab is derived from the DeepPicar project, shown in the pictures above. If you want to know more about the project, check https://github.com/mbechtel2/DeepPicar-v2.)

Part 0: Setup the project

Download the project skeleton on your Raspberry Pi 4 (not your PC) as follows.

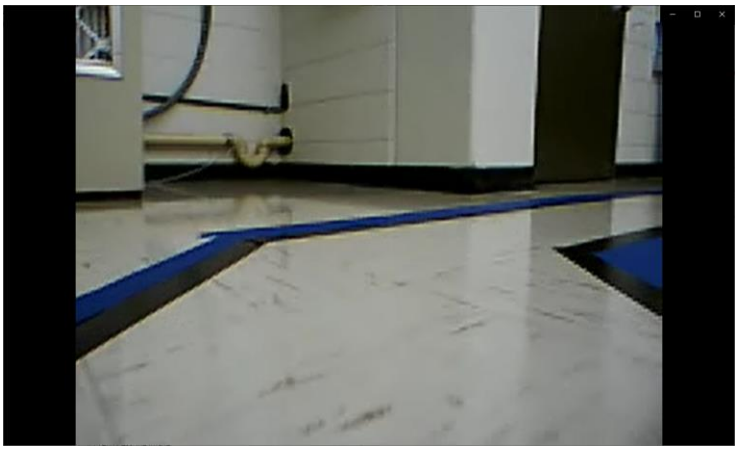
```
// Download tar from Blackboard into ~/Documents/
$ cd ~/Documents/
$ tar zxvf lab7-dnn.tar.gz
```

In the project folder, we already provided, in the folder Raspberry_Pi4, a pre-trained DNN model (model.py and model/), a sample video file (epoch-1.avi), and inference code (dnn.py). The DNN model (defined in model.py) is designed to take a camera image as input and produces a steering angle to stay in the lane as output.

You will use the same HiFive code as lab 6 for this lab, so you can use either that solution or download the lab 7 starter code from Blackboard and add your previous implementation.

The sample video file was originally created from the camera of an RC car driven by a human pilot. This video will be used as input to the DNN model instead of using an actual camera. Check the video file as follows. (You can use file browser of the pi desktop instead)

```
$ vlc epoch-1.avi
```



Part 1: Getting familiar with the TensorFlow framework (Pi 4)

In order to run a DNN-based application, we will use TensorFlow, which is a popular deep learning software framework from Google.

Using your favorite editor, open the *dnn.py* file.

In order to run any neural network, TensorFlow uses *sessions* which hold individual models and run the operations necessary for the network's architecture. In order to load our DNN model, we need to create a session and assign the model to that session:

```
#Load the model
sess = tf.InteractiveSession(config=config)
saver = tf.train.Saver()
model_load_path = "model/model.ckpt"
saver.restore(sess, model_load_path)
```

From there, we can feed input data to the loaded model to perform inferencing operations and get the control output. However, we must first collect input data and transform it such that it's compatible with the model. For this lab, we provide a *epoch-1.avi* video file and use the OpenCV image processing library for retrieving its individual frames:

```
cap = cv2.VideoCapture(vid_path)   # Open the video file
…
ret, img = cap.read()     # Retrieve the next frame from the video
```

Even though we now have data to feed to the model, we must further transform it such that it is compatible with the network architecture. If we look at the provided *model.py* file, we'll see that the network's input layer takes an input image with dimensions of 66x200x3.

```
x = tf.placeholder(tf.float32, shape=[None, 66, 200, 3],
       name="input_x")
```

Since the video, and by proxy the frames, we use for input are all 320x240x3 large, attempting to feed it to the model would generate an error. As such, we preprocess each frame such that its dimensions align with the model's input layer:

```
# Preprocess the image
img = cv2.resize(img, (200,66))
img = img / 255.
```

At this point, we have valid input data and can feed it to the model:

```
rad = model.y.eval(feed_dict={model.x: [img]})[0][0]
```

Once the inferencing operation is complete, we get an output value which represents the steering angle, in radians, the model thinks a car should use for the given input frame. How this output value is processed depends on the application and its implementation. For example, we convert the output value to degrees and then print the output and all relevant timing characteristics.

Now that we have gone over the necessary steps for loading and running a DNN, try running the *dnn.py* program and see how it performs:

```
$ python dnn.py
```

On average it should take ~21-22 ms on average to perform inferencing when the CPU is running at 1.5GHz.

Part 2: Improving network inferencing performance

Taking a closer look at the source code for *dnn.py*, you can see the following lines towards the beginning of the file:

```
#Get and set the number of cores to be used by TensorFlow
      if(len(sys.argv) > 1):
          NCPU = int(sys.argv[1])
      else:
          NCPU = 1
      config = tf.ConfigProto(intra_op_parallelism_threads=NCPU, \
                              inter_op_parallelism_threads=NCPU, \
                              allow_soft_placement=True, \
                              device_count = {'CPU': 1})
```

By default, we only have TensorFlow use a single core to perform all of the necessary inferencing operations. This can be changed by simply passing the number of cores we want to use as a command line argument when we run the program. For example, to run the DNN with all four cores available on the Pi 4 you would run the following:

```
$ python dnn.py 4
```

By doing so, we see a significant improvement to the timing performance of the DNN. At 1.5GHz, we reduce the average inferencing time by half (~11 ms).

Part 3: Communicating with HiFive

You will now combine the work done in the previous lab with the DNN model introduced in this lab. After completing this section the DNN running on the Pi should open the video file “epoch-1.avi” and output prediction angles for each video frame, these predictions should be sent to the HiFive board using the UART serial communication setup in lab 6, and finally that angle should be written back to the Raspberry Pi 4 to be displayed on a screen.

Part 3.1: Modifying the HiFive UART

You should first add your `raspberrypi_int_handler` function implementation from the previous lab into the main file `` `comm.c` `` located in HiFive directory.

Part 3.2: Adding serial communication to dnn.py

Instead of using terminals, you now run a python program on the Pi 4 to communicate with the HiFive. Your task is to extend the python program `dnn.py` that we’ve seen earlier today to be able to send the predicted degree to the /dev/ttyAMA1 serial channel. The following **pseudo-code** provides a general idea of the modifications you will need to make to `dnn.py`:

```
Import serial
Open serial connections to /dev/ttyAMA1
While(1):
  image = camera.read()
  angle = dnn_inference(image)
  Write ‘angle’ to /dev/ttyAMA1
  Wait_till_next_period()
Close serial connection
```

To achieve the functionality from above, you need to use Python's pySerial API which can be included by importing the serial package:

```
import serial
```

Once the serial package has been imported, you should create a channel for writing the degree value to the HiFive1 over /dev/ttyAMA1. Note that the channel should be opened with the baudrate of 115200 bps.

```
ser1 = serial.Serial(...)
```

The angles received from the DNN as it processes frames can then be sent to the HiFive board by using the serial `write()` function:

```
ser1.write(...)
```

However, `write(...)` requires a byte value while the angle produced by the DNN is a `float32` value, so you will have to convert the angle data in order to send it to the HiFive1. You may also find it helpful to cast the prediction angle from a `float32` to a different type before transmitting in the python code. Finally, recall that the `readline(...)` function needs a closing newline character (`\n`) that you will need to send.

```
bytes(degree)
```

Finally, after all of the frames are processed, the serial connection needs to be closed by invoking the serial `close()` function:

```
ser1.close()
```

To demo your code, you will need to open a serial screen, on the Pi, on ‘dev/ttyAMA2’ to receive the predicted angle being written back to the Pi from the HiFive board.

For submission, turn in a compressed archive like the previous labs which contains your changes to `dnn.py` in the Raspberry Pi directory – you don’t need to submit your implementation of `raspberrypi_int_`handler from lab 6.

Appendix A: Installing TensorFlow on your own Pi 4.

While TensorFlow is already installed on the Raspberry Pi 4's in the lab, you can install it on your own machines as well. This can be done by using the pip package manager for Python modules. If Python and/or pip aren't already installed, they can be with the following commands:

```
$ sudo apt-get install python-dev
$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
$ sudo python get-pip.py
```

Once installed, TensorFlow can be installed with the following commands:

```
$ sudo apt-get install libhdf5-dev
$ sudo pip install --no-cache-dir tensorflow
```

Note that the libhdf5-dev library is needed for another module TensorFlow uses, h5py. Lastly, install OpenCV packages for image processing.

```
$ sudo apt-get install python-opencv
```

To use tensorflow in python3, do the following (optional).

```
$ sudo pip3 install --no-cache-dir tensorflow
$ sudo apt-get install python3-opencv
```