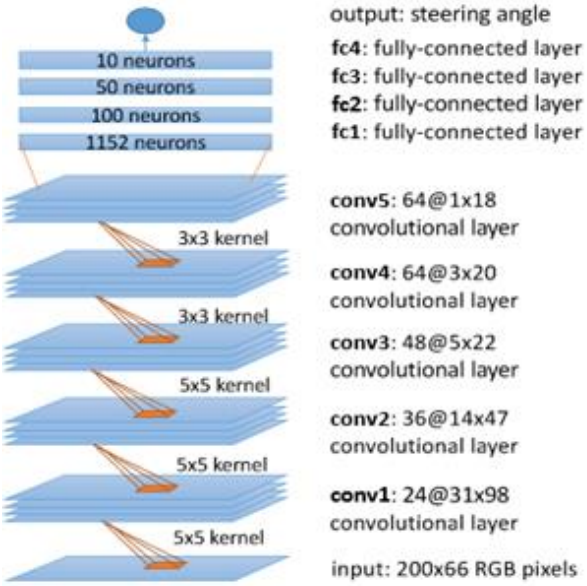
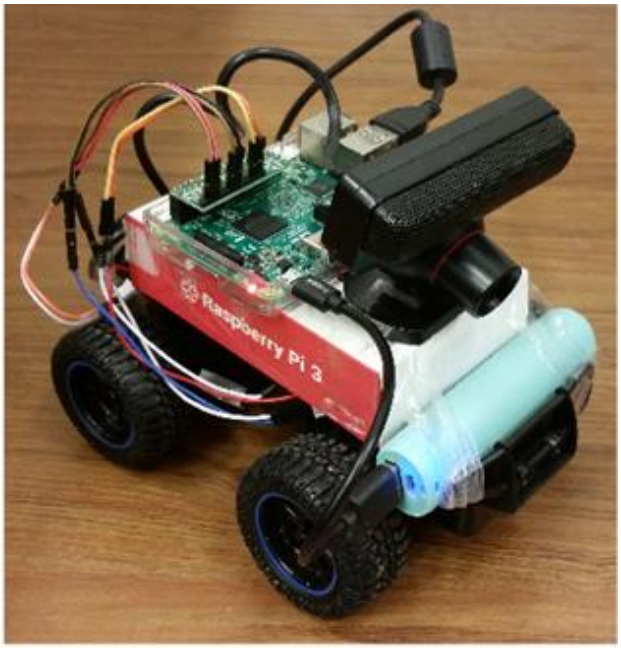
EECS 388 Lab #8

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.

Part 0: Setup the project

‘

Do this task in your Raspberry Pi4 (Not in the linux PC).

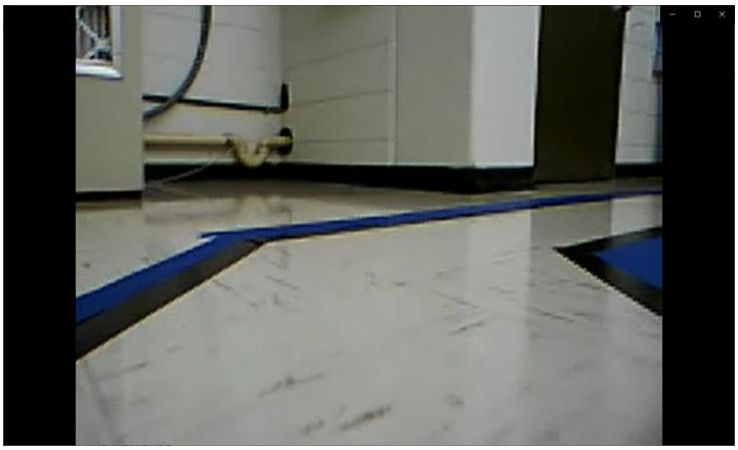
First sign in to the Raspberry Pi account with login ID and password.
Sign in to your Canvas account and go to the EECS 388 Embedded Systems Lab section. Inside the “Files” section, go to the “Source Codes” folder. Click on the Lab08-dnn.tar.gz file and download it to your Raspberry Pi. Extract the project folder.

In the project folder, we already provide a pre-trained DNN model (model.py and model/), a sample video file (epoch-1.avi), and interference 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.

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

In order to run a DNN-based application, we will useTensorFlow, 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).

Today's task :

Run the DNN inference code for CPU core numbers 1, 2 , 3 and 4.

Your task is to measure the performance---especially *mean*, which represents the average, and *max*, which represents the worst-case---of the inferencing operations while varying the number of CPU cores being used from 1 to 4.

Prepare a table showing the CPU core numbers, the mean and max time of inferencing operations (shown below) and write a comment on the result. (for example: how the inferencing operation time is varying with the CPU core, you can show your result using a graph as well.)

CPU Core	Mean time	Max time
1		
2		
3		
4		

For demo, you need to show the procedure and results to your GTA's.
You need to prepare a report including ONLY the table and the comment, and submit it on Canvas as a PDF or doc file. (screenshots/picture will NOT be allowed as submission).

The following section is just for your information, not needed for your lab tasks.

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
```