Self-Driving Car using Convolutional Neural Network

Chris Winsor

2/8/2019

We explore Convolutional Neural Networks and the Keras/Tensorflow library as applied to whole-image classification. Our work is performed using real-world data. Classification is done in real time.

The task at hand (a contrived project) is the development of a self-driving toy car. The car has left/right/center steering and an on-board Raspberry Pi with camera. A CNN on the Pi is taught to steer the car by classifying the camera image into left/right/center steering values. The CNN is taught by observing an “expert” drive around a variety of tracks.

The project developed in conjunction with the MetroWest Boston Developers Machine Learning Group from Framingham MA.

This document includes:

- Overview of the car, track and general approach
- Dataset construction, CNN architecture, training, testing
- Hardware: Interfacing the Raspberry Pi, the camera, the RC controller and the car

Table of Contents

Table of Contents

References and Links

Source Code and Dataset:

- https://github.com/cwinsor/metrowest_scikit_tensorflow_cnn_car

MetroWest Boston Developers Machine Learning Group:

- Meetup: Metrowest Boston Developers Machine Learning Group
- GitHub: https://github.com/geneostrat/Metrowest-Developers-Machine-Learning-Group
- Original raw data: https://github.com/geneostrat/TrainingData.git

Book:

- Hands-On Machine Learning with Scikit-Learn & TensorFlow, Geron, O'Reilly, First Edition

Keras Library

- https://keras.io/
- https://github.com/keras-team/keras/blob/master/examples/README.md

TensorFlow (non-GPU version):

- https://www.tensorflow.org/

Raspberry Pi

- https://www.raspberrypi.org/

Overview

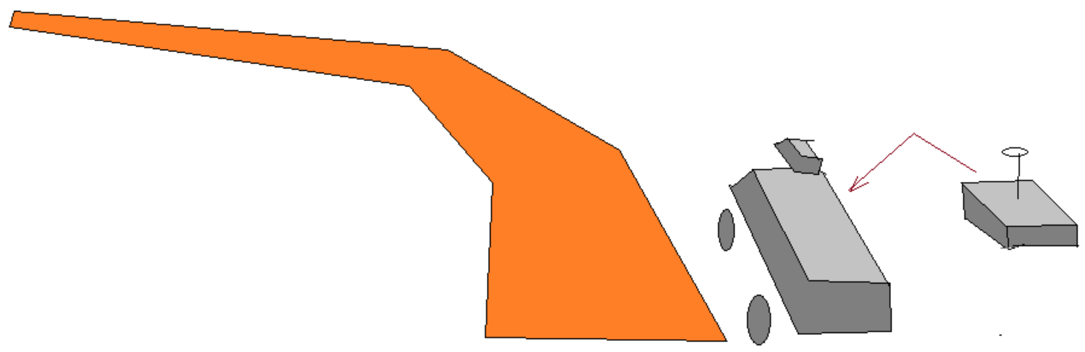
The Car

The car was the “RC Trucks” by New Bright ($10 Walmart). It includes a Handheld Controller with forward/backward and left/right (non-proportional) controls. The communication link between controller and car is 49mhz.

NEW BRIGHT R/C
RC TRUCKS
Full Function Radio Control
Forward/Reverse Driving
Left/Right Steering
Instructions On Package Bottom
BATTERY REQUIREMENTS: (not included)
VEHICLE: 3 "AA" Batteries.
TRANSMITTER: 2 "AA" Batteries.
MADE IN CHINA
NEW BRIGHT INDUSTRIAL CO., LTD.
NEW BRIGHT BUILDING, 11 SHEUNG YUET ROAD
KOWLOON BAY, KOWLOON, HONG KONG.

The Track

The “track” was a piece of orange tape on a white carpet floor.



The Raspberry Pi

For on-board processing we used a Raspberry PI 2 Model B with optional Camera running the Raspbian (Ubuntu Linux). The Pi was zip-tied to the car with camera facing forward.

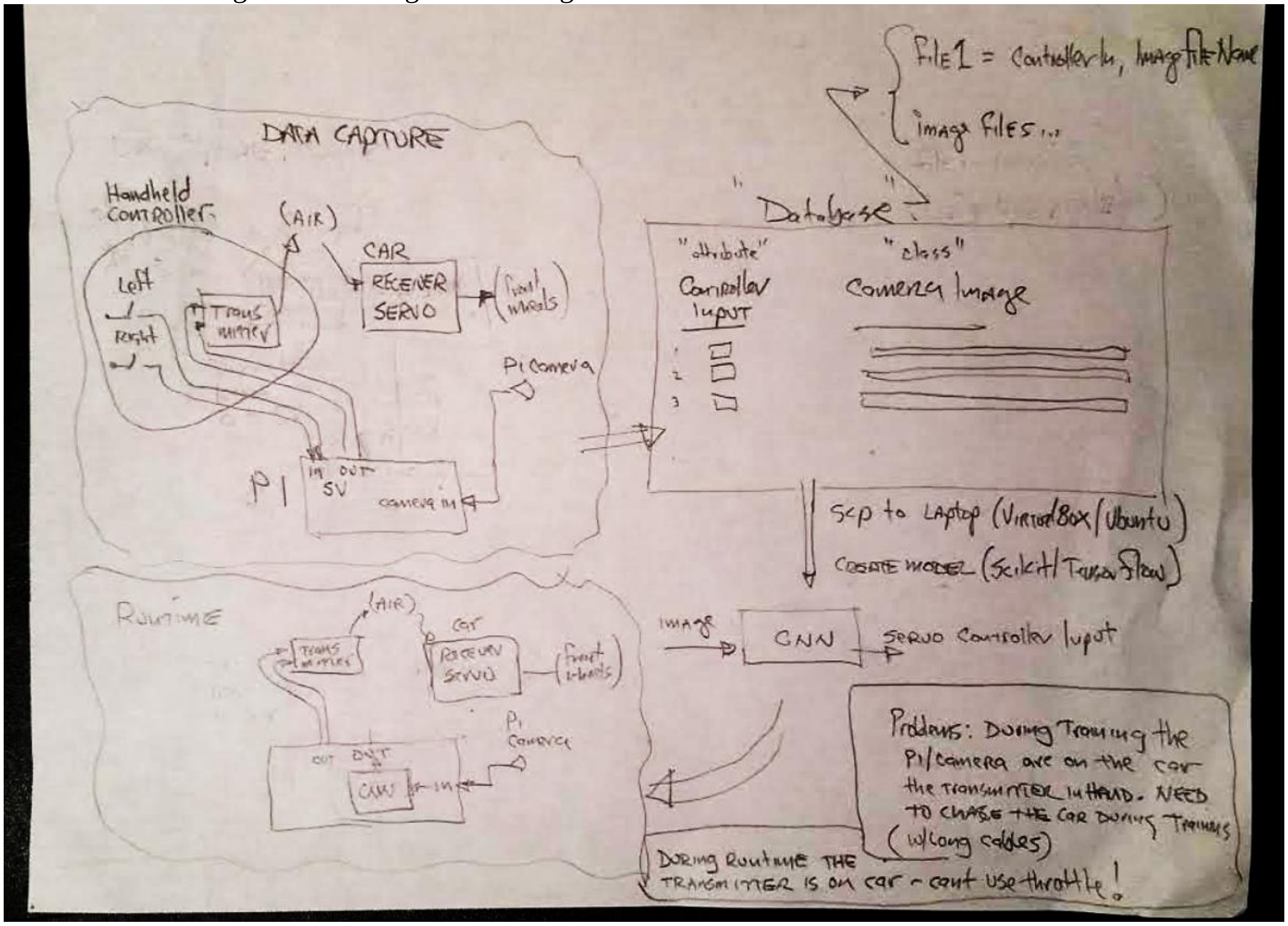
Software

We used the Keras/TensorFlow libraries from Google. Keras is a neural network Python library which simplifies the steps needed to specify, train, and run a CNN. TensorFlow is framework which allows computation to be distributed across CPUs or GPUs. It is particularly capable for graph-based structures like those of a neural network.

For the Pi there are two small python applications - the first captured training data, the second applied the trained CNN to drive the car. We use the standard Raspbian Linux on the Raspberry PI.

General Approach

In the “Data Capture” phase directional control signals and images from the camera were captured as a user drove the car. This data was then copied to the development workstation where the Keras/TensorFlow library was used to train a Convolutional Neural Network. The output of training was a file with structure and weights for the CNN. The file allows the CNN to be re-instantiated on the Pi with parameter weights applied. The Pi was then able to use the CNN to convert real-time image from the camera to generate left/right control signals to drive the car.


DATA CAPTURE
Handheld CONTROLLER
Left
Right
Trans mitter
(AIR)
CAR
RECEIVER SERVO
front wheels
Pi Camera
Pi
IN OUT
SV
camera IN
File1 = ControllerIn, ImageFileName
image files...
"Database"
"attribute"
Controller Input
1
2
3
"class"
Camera Image
scp to Laptop (VirtualBox/Ubuntu)
CREATE MODEL (Scikit/TensorFlow)
image
CNN
SERVO Controller Input
Runtime
(AIR)
car
Trans mitter
RECEIVER SERVO
front wheels
Pi Camera
OUT
OUT
CNN
IN
Problems: During Training the Pi/camera are on the car the transmitter in HAND. NEED TO CHASE THE CAR DURING TRAINING (w/Long cables)
DURING Runtime THE TRANSMITTER IS ON car - cant use throttle!

Some tactical considerations:
During training the Pi needs to be two places at the same time - on the car taking pictures, and near the Handheld Controller observing signals into the DK2970. To solve this, we will use a cable, so the Pi can be on the car but can observe the signals going into the DK2970. There is no problem during Runtime since the Handheld Controller can be relocated to the car (no user is required) so it is co-resident with the Pi.

The Dataset

Overview

The dataset is a set of samples consisting of a camera image and steering value. The primary feature in the image is a piece of red tape which identifies the centerline of the track. The background of the track is white carpet. The steering values are categorical (left/straight/right).

The dataset consists of 18921 samples cut into training subset of 15136 and test subset of 3785.

Images are 45h by 90w using RGB encoding. This is kept as a 4-dimensional NumPy array of numpy.uint8. The dimensions of this array are [N][45][90][3]. The first dimension is image number. The next two dimensions are height, width. The final dimension is color.

Steering values are categorical with 1=left, 3=straight 2=right. The data is kept as a 1-dimensional NumPy array of numpy.uint8. The array is size [N] which is the direction the car is being steered at the time of the correspondingly-numbered image.

The structure of the dataset is:

- "DESCR" - an overview description of the dataset.
- "images_train"   - training images
- "images_test"   - testing image
- "steering_train" - training steering values
- "steering_test" - testing steering values
- "target_names"   - a list of values of the steering class [1, 3, 2]

Creating the Dataset

This section describes the process used to create the Dataset. This section includes reference to the source of the original Raw Data, steps to preprocess the Raw Data, and steps to create the Dataset.

The Raw Data

The raw data (I will refer to as “Geneostrat Training Data”) was established by Gene Olafsen over a several-week period in late 2018 and early 2019. The raw data consists of images and corresponding steering values from a similar RC car using a front-mounted Raspberry PI camera.

Geneostrat Training Data is available on GitHub:
https://github.com/geneostrat/TrainingData
commit 55a8a9b8af8f57920217d24c3a7499b764abae35
Author: GeneO <engineering@strattonassociates.com>
Date: Tue Jan 1 21:34:01 2019 +0000

The raw data was captured using various track configurations. They include:

- mixed turns and straights (a wandering oval)

- fixed radius circular loops
- straight sections

Some tracks have poor lighting (entirely black) or are not representative (use a double-line).

Clockwise and counter-clockwise runs are included for the fixed-radius and mixed oval. The intent is that all degrees of curves (and straights) would be represented in the data.

The attributes are the pixels within the images. Images are 180 x 90 RGB.

The class is numeric, a value reflecting the position of the steering. Values can range between 273 and 528 where 400 is "straight ahead", lower values are left, and higher values are right.

Preprocessing the Raw Data

Removing Poor Quality and Non-Representative Samples

We first eliminated runs with poor quality (lighting) and those using dual-line tracks. This left us with 24 runs:

"wandering oval":
    "121",   "122",   "124",   "125",
"fixed radius":
    "R18CCW",   "R18CCW_V",   "R18CW",   "R18CW_V",
    "R20CCW",   "R20CCW_V",   "R20CW",   "R20CW_V",
    "R21CCW",   "R21CCW_V",   "R21CW",   "R21CW_V",
    "R25CCW",   "R25CCW_V",   "R25CW",   "R25CW_V",
"straight"
    "STR1",   "STR1_V",   "STR2",   "STR2_V"

Each run had between 369 and 1486 images. The total images were 18922.

Converting Class Variable

Although the Geneostrat car uses the same Raspberry PI there are minor differences in the car and controller. Specifically - the Geneostrat car uses numeric steering values while our car uses 3 discrete directions. We thus need to convert the class variable in in the Raw Data from numeric to a categorical class. To do this we assess the raw data with the intent of establishing thresholds.

We first considered the "CW", "CCW" with 18" to 25" radius:

- CCW was 8 runs 6348 images mean=350.3 std_dev=21.5
- CW was 8 runs 5250 images mean=429.2 std_dev=21.2

We observe 20% more CCW (left turning) samples than CW (right turning). The mean of the left turns was 50 points off the center of 400, while the mean of the right turns was 30 points off the center. Not only are left hand turns are more frequent, but the value associated with those is stronger.

We then consider the "STR" cases:

- STR was 4 runs 2427 images mean=399.1 std_dev=10.6

We observe the “straight” is closely centered around the value of 400. There are significantly fewer “straight” samples than either CW or CCW.

We then considered the “wandering oval”

- WO was 4 runs 4897 images mean=393.7 std_dev=31.7

We observe the “wandering oval” is a broader distribution than even the R18-25, and slightly favored left turns.

Based on the investigation we established thresholds for “left”, “straight”, “right” (our categorical classes). The thresholds chosen were 390 and 402. With these thresholds we have:

```
left =    7644
straight = 6265
right =   5012
```

Our thresholded data has 40% more left turns than right - this reflects the left turn bias in the raw data.

Preparing the Dataset

We then prepared our Dataset.   This involves:

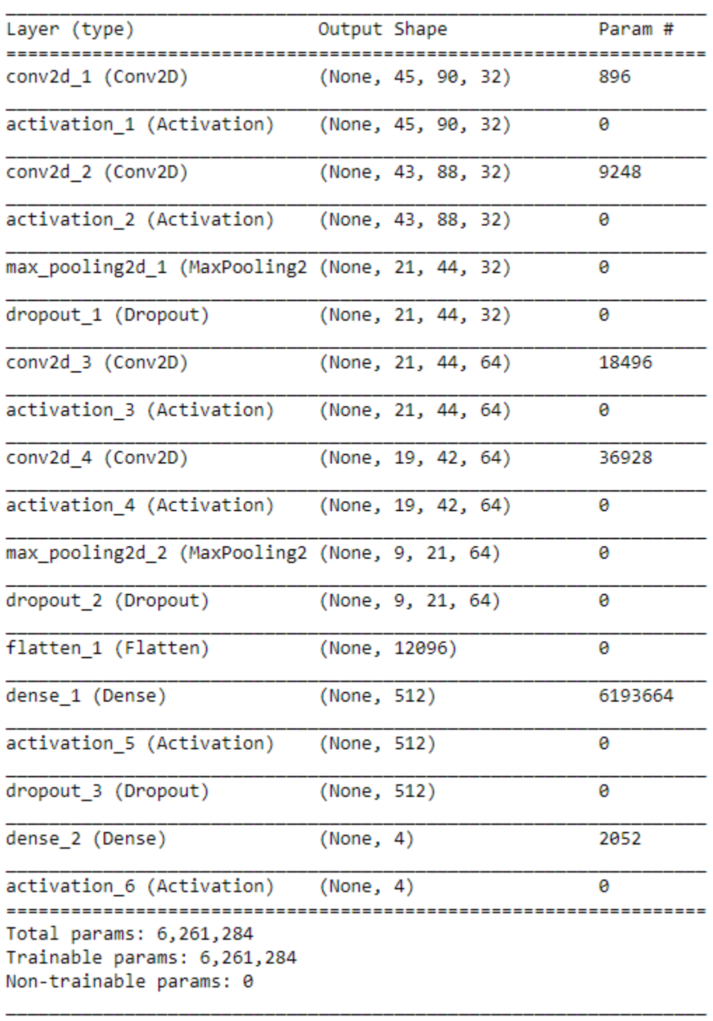
- reading raw image and steering data
- applying steering threshold (converting steering to categorical)
- down sampling from 90x180 to 45x90
- shuffling
- segmenting into training/testing subsets
- saving to pickle file

CNN Structure

We base our CNN structure on the MNIST example from Keras. MNIST dataset is like ours - difference include the number of bits in the image and RGB vs black/white image content.
References:
https://github.com/keras-team/keras/tree/master/examples
https://github.com/keras-team/keras/blob/master/examples/mnist_mlp.py

	MNIST	Self-Driving Car
Goal	Full-image classification	Full-image classification
Features	Multiple lines with shape and intersections as significant	Single line with shape and placement as significant
Image encoding	bi-tonal - black lines on white background.	bi-tonal (red line on white background)
Target Class	Categorical (10 classes)	Categorical (3 classes)
Input Image size	28x28 black/white	45x90 RGB

Our CNN is structure is:



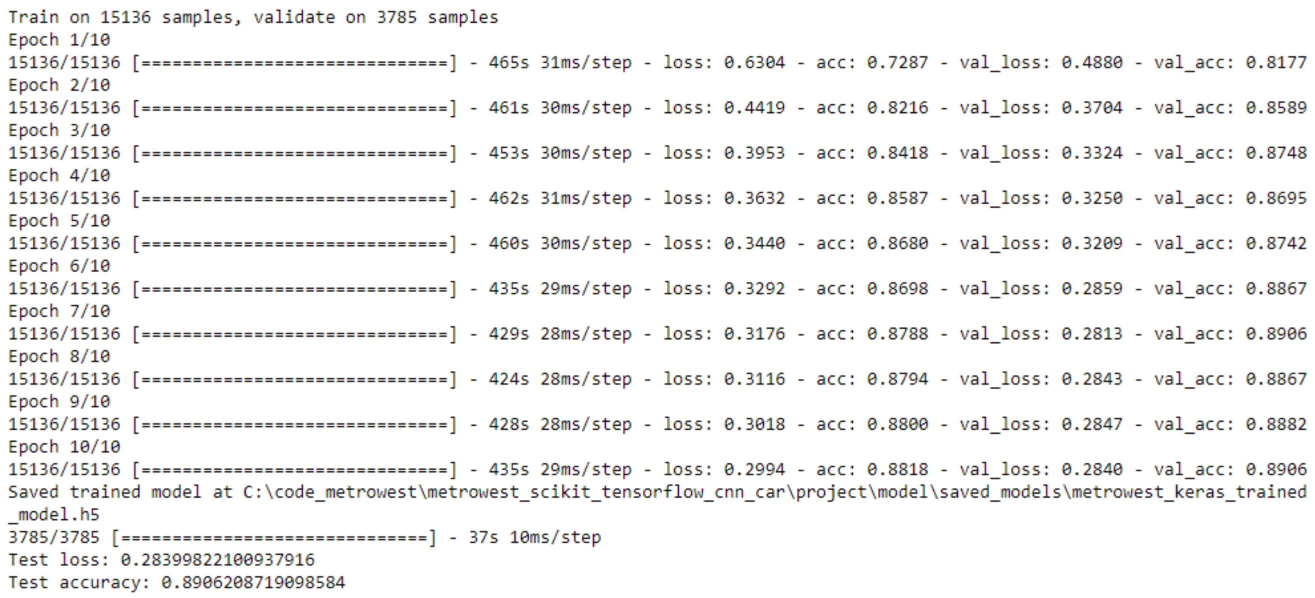
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 45, 90, 32)        896
_________________________________________________________________
activation_1 (Activation)    (None, 45, 90, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 43, 88, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 43, 88, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 21, 44, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 21, 44, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 21, 44, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 21, 44, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 19, 42, 64)        36928
_________________________________________________________________
activation_4 (Activation)    (None, 19, 42, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 9, 21, 64)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 9, 21, 64)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 12096)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               6193664
_________________________________________________________________
activation_5 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 2052
_________________________________________________________________
activation_6 (Activation)    (None, 4)                 0
=================================================================
Total params: 6,261,284
Trainable params: 6,261,284
Non-trainable params: 0
_________________________________________________________________
```

Performance

We achieved 89% accuracy with 10 epoch training set. Training took about 2 hours using a typical laptop. The biggest factor in training time was number of parameters.

Physical memory on the Pi drove the decision to down-sample the image. The original raw images of 90x180 were 16kbytes, and the resulting model would not fit into the Pi's physical memory due to the number of parameters. When down-sampled to 45x90 the result is a 4kbyte image and corresponding reduced model size.

The net result is the Pi's memory limitation forced a reduction in the model size. Thus, the model could be easily trained on a regular laptop (no cloud or GPU required).



```
Train on 15136 samples, validate on 3785 samples
Epoch 1/10
15136/15136 [==============================] - 465s 31ms/step - loss: 0.6304 - acc: 0.7287 - val_loss: 0.4880 - val_acc: 0.8177
Epoch 2/10
15136/15136 [==============================] - 461s 30ms/step - loss: 0.4419 - acc: 0.8216 - val_loss: 0.3704 - val_acc: 0.8589
Epoch 3/10
15136/15136 [==============================] - 453s 30ms/step - loss: 0.3953 - acc: 0.8418 - val_loss: 0.3324 - val_acc: 0.8748
Epoch 4/10
15136/15136 [==============================] - 462s 31ms/step - loss: 0.3632 - acc: 0.8587 - val_loss: 0.3250 - val_acc: 0.8695
Epoch 5/10
15136/15136 [==============================] - 460s 30ms/step - loss: 0.3440 - acc: 0.8680 - val_loss: 0.3209 - val_acc: 0.8742
Epoch 6/10
15136/15136 [==============================] - 435s 29ms/step - loss: 0.3292 - acc: 0.8698 - val_loss: 0.2859 - val_acc: 0.8867
Epoch 7/10
15136/15136 [==============================] - 429s 28ms/step - loss: 0.3176 - acc: 0.8788 - val_loss: 0.2813 - val_acc: 0.8906
Epoch 8/10
15136/15136 [==============================] - 424s 28ms/step - loss: 0.3116 - acc: 0.8794 - val_loss: 0.2843 - val_acc: 0.8867
Epoch 9/10
15136/15136 [==============================] - 428s 28ms/step - loss: 0.3018 - acc: 0.8800 - val_loss: 0.2847 - val_acc: 0.8882
Epoch 10/10
15136/15136 [==============================] - 435s 29ms/step - loss: 0.2994 - acc: 0.8818 - val_loss: 0.2840 - val_acc: 0.8906
Saved trained model at C:\code_metrowest\metrowest_scikit_tensorflow_cnn_car\project\model\saved_models\metrowest_keras_trained
_model.h5
3785/3785 [==============================] - 37s 10ms/step
Test loss: 0.28399822100937916
Test accuracy: 0.8906208719098584
```

Hardware

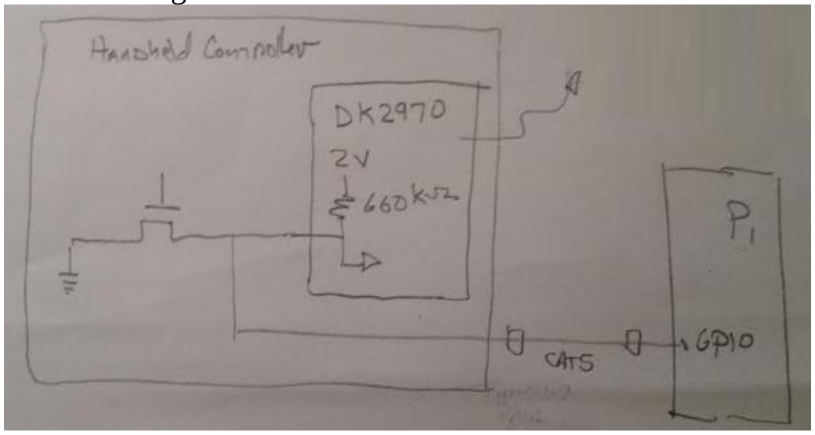
Hacking the Handheld Controller

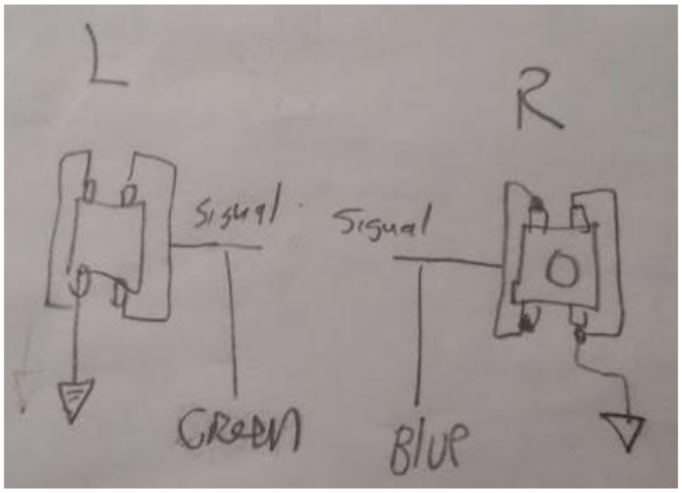
Inside the controller is a small circuit board with four button switches (left/right/forward/reverse), a transmitter chip (DK2970), and analog components to support the antenna. The chip handles everything - taking in the 4 control signals and outputting the 49mnz carrier. The control signal inputs are 2.1 volt “open drain” - a push of the button grounds the signal. The pullup is internal to the chip and measured at 660kohm. We are fortunate in that signal levels are compatible with the Pi GPIO.

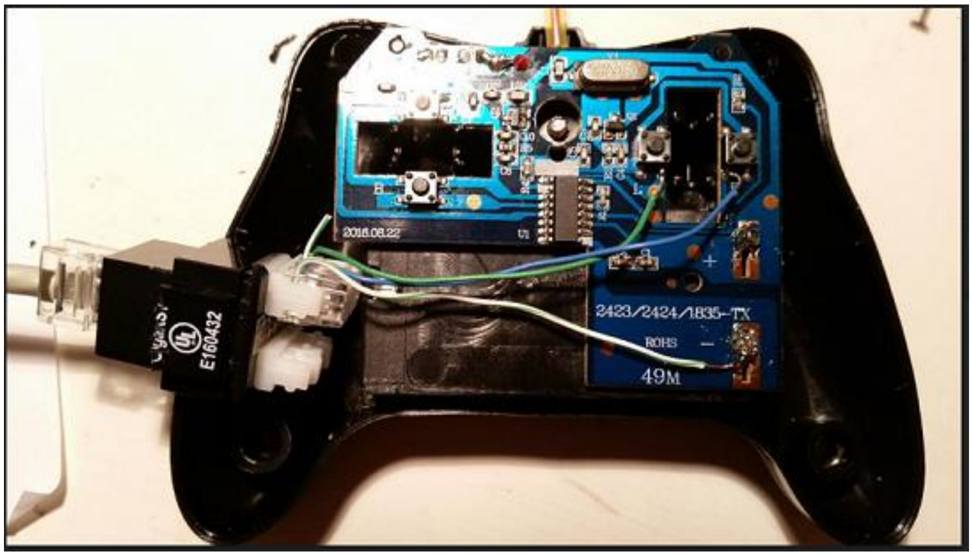
During training the Pi wants to receive the left/right signals. During runtime the Pi will be driving them. The brute force approach (shown in the “Architecture Diagram”) involves breaking the signal between the button and the DK2970, having the Pi intercept, observe, and output the signals during training. In theory this is correct, it would require 2 wires (a 'receive' and transmit') to/from the Pi for Left and 2 for Right (4 total).
A simpler approach was used leveraging the open-drain signal of this controller. During training the Pi simply observes the Handheld Controller signals - the push buttons on the Handheld Controller ground the wire to produce the signal. During runtime the handheld still provides the pullup, but the Pi will ground the wire to produce the signal. This means only 1 wire is needed between Handheld Controller and Pi for each of Left and Right (2 total).

Noise is a consideration especially considering the weak 660Kohm pullup and the length of the cable. We used CAT5 twisted pair to reduce noise exposure.

Finally - we added a CAT5 connector - this allows substituting a shorter cable during runtime when the Handheld Controller will be riding on the car near the Pi.





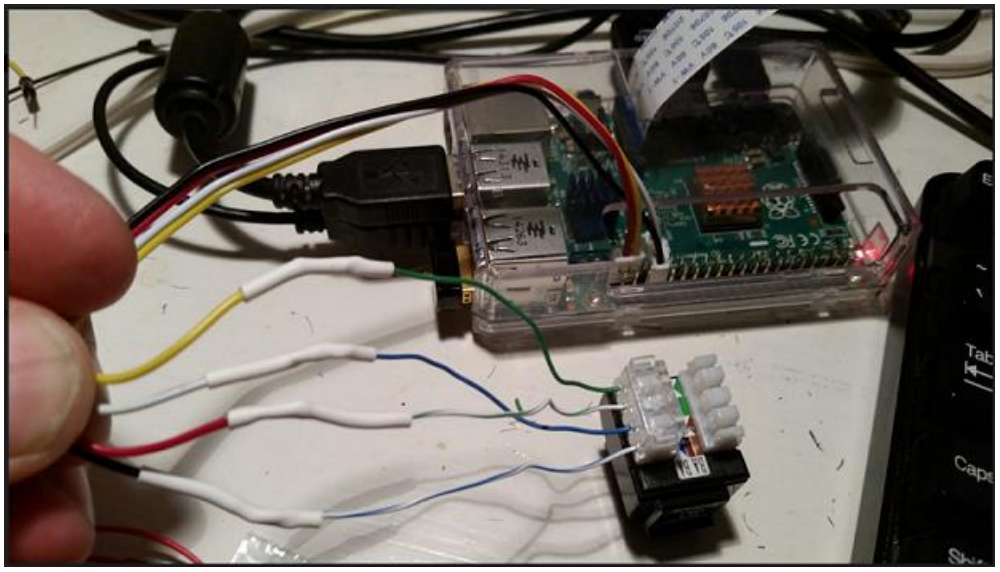


GPIO connector and Software

For physical wiring we chose GPIOs 26 and 16 and again used a CAT5 connector. A reference for GPIO pinning on the Pi2-B (that is actually *correct*) is https://www.youtube.com/watch?v=6PuK9fh3aL8

Code to verify signaling between Pi GPIO and Handheld Controller is in Appendix X.
We use the "gpiozero' python library https://gpiozero.readthedocs.io/en/stable/index.html .

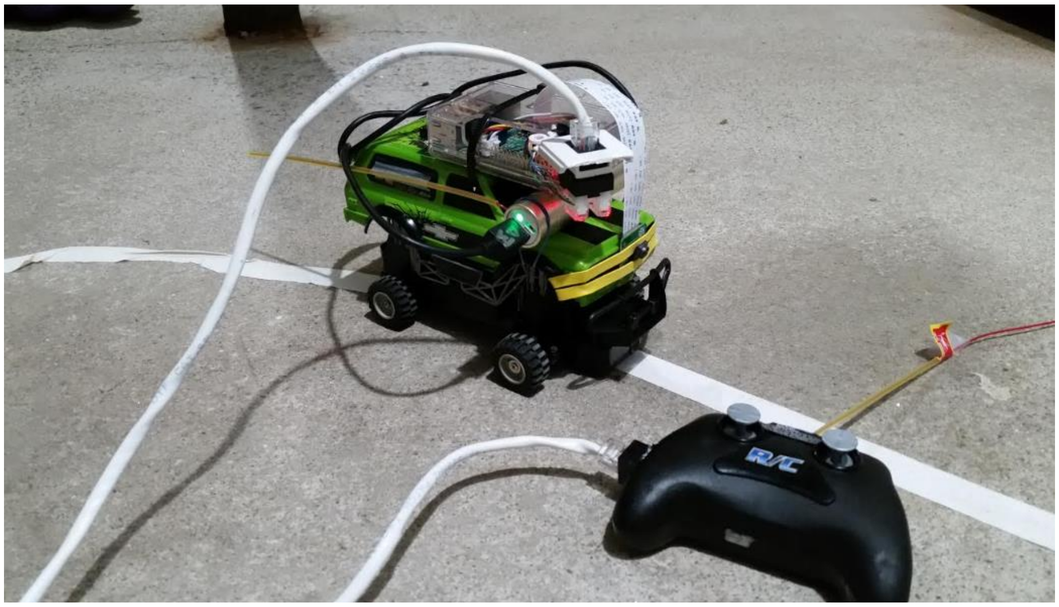
With the above wiring and software we demonstrated the Pi can capture the signal and camera training data and can drive the left/right signals to turn the car. Now it is time to capture data!



Car Assembly
The Pi was zip-tied to the top of the car. A USB battery was used to power the Pi. The camera was rubber-banded to the front grill. A CAT5 provides the control signals to the Pi for observation during training.

We laid out an "Oval" track. We ran the car around the track in clockwise and counter-clockwise directions - a total of 11 loops. Each loop is about a minute and the capture rate is 2/second.

Raw data and Python code are available on the GitHub.



Appendix 1: Environment Setup (Instructions)

The following steps explain how to set up and run the environment. The environment consists of a Development Workstation and a Raspberry Pi.

The Development Workstation

The instructions assume Windows 10 with Visual Studio Code and Jupyter Notebook. Linux and IOS should work but are not tested.

- git clone https://github.com/cwinsor/metrowest_scikit_tensorflow_cnn_car
- cd metrowest_scikit_tensorflow_cnn_car

One-time environment setup

This establishes virtual environment and downloads necessary modules. From PowerShell run:

- setup_onetime_00_restore_libraries.ps1

Every-time environment setup

These activate the virtual environment, adds local /lib to the path, starts Jupyter notebook, and starts Visual Studio Code IDE.

1. setup_everytime_00_activate_env
2. setup_everytime_01_add_cwd_to_path
3. setup_everytime_02_start_jupyter_notebook
4. setup_everytime_03_start_code

Creating and Using the Dataset

Files in the *project/lib* are modules to create and use the dataset. Python (.py) are used as library modules and Jupyter Notebooks are the tests/demonstrate the use.

Using the dataset: A Jupyter Notebook shows how a user would load the dataset:

- step_05_user_load_the_dataset .ipynb

Architecting the dataset

This starts with the raw “Geneostrat” data files and prepares the “pickle” file which is the dataset. Users should not typically have to do this.

- step_04 architect_the_dataset.ipynb

Module-level tests

Additional notebooks are provided which test the file load and display of images

- step_01_test_file_io.ipynb
- step_02_test_display.ipynb
- step_03_preprocess.ipynb

Building the (CNN) Model

Files in *project/model* allow creating and training the Keras/TensorFlow CNN that is the heart of the project. This notebook constructs the model, loads the data, trains the model, and saves the trained model (structure and weights) to a .h5 file.

Using an image of 45x90, running batch size 32, 10 epochs on a I75500 2.4ghz 16GB mem Win10 (GPUs available but not used), takes about 1hr/epoch = 10 hours and achieves an accuracy of <TBD???>

- Learn_to_Drive.ipynb

The Raspberry PI

Raspberry Pi 3 or Pi 2 are assumed for the car. The files are in project/rasp_pi

- SSH to the PI.
- git clone https://github.com/cwinsor/metrowest_scikit_tensorflow_cnn_car
- cd metrowest_scikit_tensorflow_cnn_car/project/rasp_pi

One-time environment setup

(NOTE these are in the rasp_pi directory and are different than the setup files for the workstation above). The one-time setup establishes virtual environment and downloads necessary modules. From bash shell run:

- source setup_onetime_pi_00_restore_libraries.sh

Every-time environment setup

These activate the virtual environment and add local /lib to the path.

- source setup_everytime_pi_00_activate_env.sh
- source setup_everytime_pi_01_add_lib_to_path.sh

Running the Car

This python module pulls in the CNN model, starts capturing images to steer the car.

- python3 self_driving_car_drive.py

Capturing Raw Data and Test

It should not be necessary for users to capture raw data (we leverage raw data captured in “Geneostrat”) but a file is provided for this purpose if needed. Also – a test file for debugging the PI and GPIO connections to the controller.

- python3 self_driving_car_capture.py
- python3 self_driving_car_test.py

Appendix 2 (Sample Code)

- The following code was used when hacking the Handheld Controller and Pi to demonstrate operation of the physical wiring and connections between Handheld Controller and Pi GPIO via the CAT5 cable. This is a subset of the full code which is available on GitHub.
-
- The “train” switch sets the mode of operation. In “train” mode the Pi observes GPIOs and prints a left/right message when the Handheld Controller left/right buttons are pushed. In “non-train” (driving) mode the Pi will output left and right signals on a 1 second period.   If the car is powered the front wheels will follow the Handheld Controller (training) or the Pi (driving). Camera images are also captured in “train” mode.

```
from gpiozero import InputDevice
from gpiozero import OutputDevice
from picamera import PiCamera
from time import sleep

train = True

pin_left  = 26
pin_right = 13

if train == True:

        pin_l = InputDevice(pin_left,True)
        pin_r = InputDevice(pin_right,True)

        camera = PiCamera()
        camera.start_preview()
        sleep(5)
        i = 0
        while True:
                if pin_l.value:
                        print("Left")
                if pin_r.value:
pi@raspberrypi2:~/tempdir$ more temp.py
from gpiozero import InputDevice
from gpiozero import OutputDevice
from picamera import PiCamera
from time import sleep

train = True

pin_left  = 26
pin_right = 13

if train == True:

        pin_l = InputDevice(pin_left,True)
        pin_r = InputDevice(pin_right,True)

        camera = PiCamera()
        camera.start_preview()
        sleep(5)
        i = 0
        while True:
                if pin_l.value:
                        print("Left")
                if pin_r.value:
                        print("Right")

                camera.capture('/home/pi/tempdir/pictures/image%s.jpg' % i)
                i = i + 1
                sleep(1)
        camera.stop_preview()


if train == False:

        pin_l = OutputDevice(pin_left,True,True)
        pin_r = OutputDevice(pin_right,True,True)

        while True:
                print("center")
                pin_l.on()
                pin_r.on()
                sleep(1)

                print("right")
                pin_l.on()
                pin_r.off()
                sleep(1)

                print("center")
                pin_l.on()
                pin_r.on()
                sleep(1)

                print("left")
                pin_l.off()
                pin_r.on()
                sleep(1)
```