Self-Driving Car using Convolutional Neural Network

Chris Winsor

9/1/2018

The experiment applies a Convolutional Neural Network (CNN) to the task of making a self-driving car. The car is taught to follow a track which defined as a piece of orange tape on a white carpet floor. Data consists of left/right control signals of an operator as the car is driven and images from a forward-mounted camera. The CNN is trained to predict left/right control signal from the camera image. The CNN is then used to control the car without user input. The project demonstrates the use of a CNN in a real-world application.

The software library is Python/Scikit-Learn/TensorFlow. The hardware platform is Raspberry Pi/Raspbian.

This document includes:

- Overview of the car, track and general approach
- Hardware: interfacing the RaspberryPi to the RC controller and Pi Camera
- Software: CNN architecture, database structure, files, and approach to data collection

References and Links

Source Code and Processed (categorical) Data:

- 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
- Raw data (proportional control): https://github.com/geneostrat/TrainingData.git

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

<reference to the 2 datasets, and to the examples>

Keras:

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

Tensorflow (non-GPU version):

Setting Up the Environment

The instructions assume Windows PowerShell. Linux and IOS should work but are not tested.

1. git clone https://github.com/cwinsor/metrowest_scikit_tensorflow_cnn_car
2. cd metrowest_scikit_tensorflow_cnn_car
3. setup_onetime_00_restore_libraries.ps1
4. setup_everytime # this will activate the environment, add to PATH variable, start Jupyter Notebook, start Visual Studio Coe

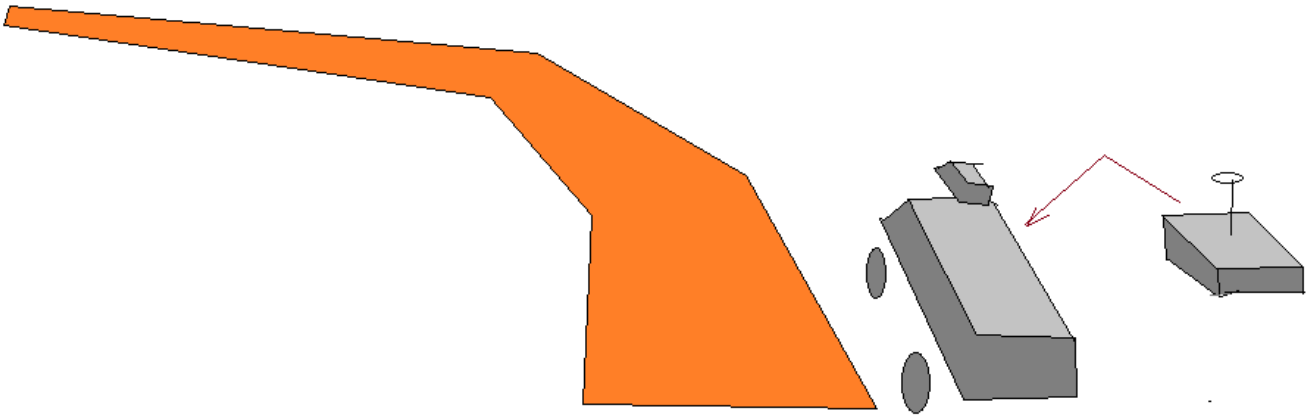
Overview

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.

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

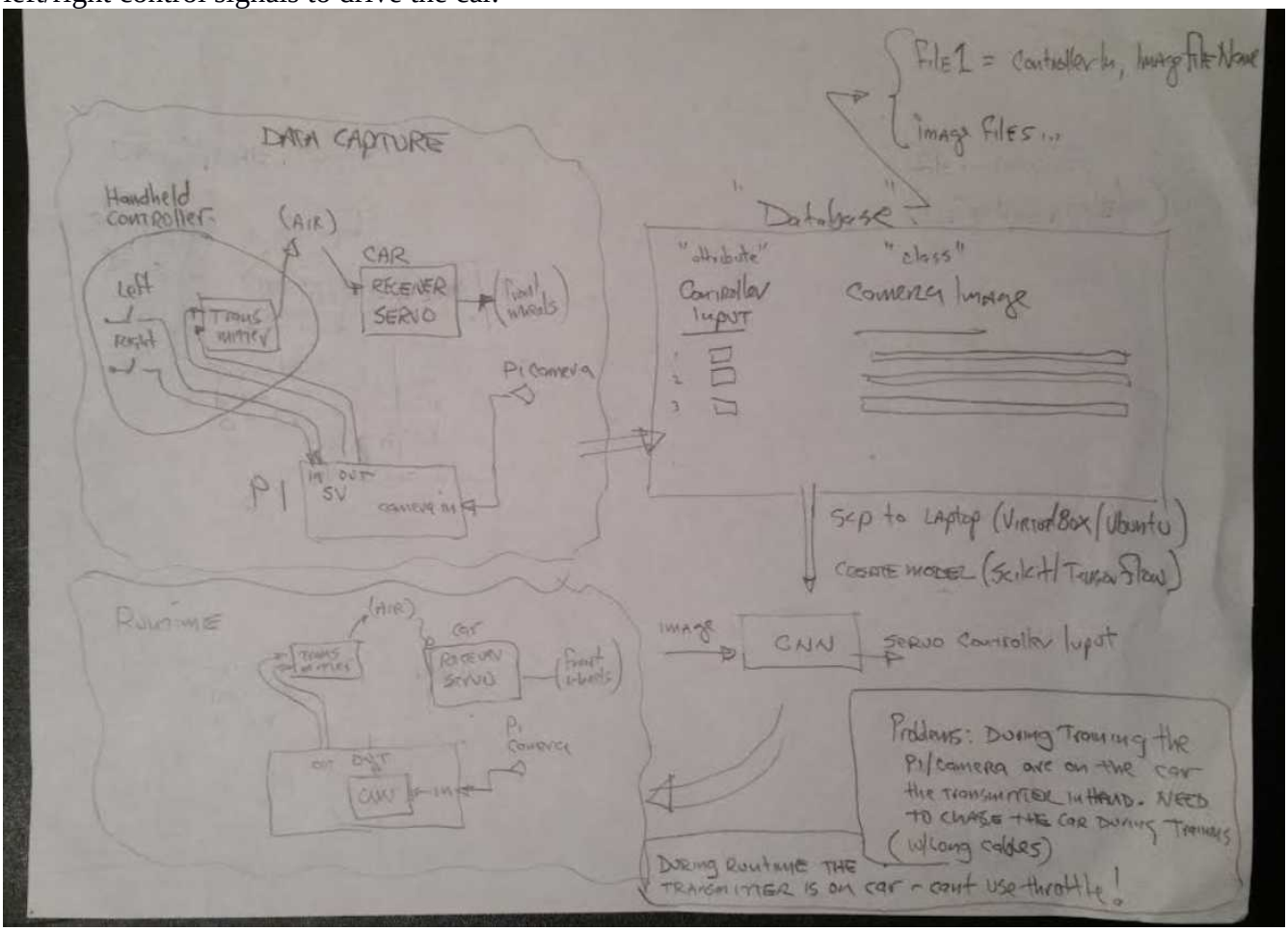
We used the Keras/Tensorflow libraries from Google for the CNN. Keras is a neural network Python library which simplifies the steps needed to specifify, train, and run a CNN. Tensorflow is framework which allows computation to be distributed across CPUs or GPUs. It is particularly capable for graph-

based structures like those of a neural network. Training was performed on the Google Cloud.

For the Pi, two small python applications - the first captured training data, the second applied the trained CNN to drive the car. We use the standard Raspibian Linux on the Raspberry PI with the (????) library.

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 a cloud compute farm where the Scikit/TensorFlow library was used to train a Convolutional Neural Network (“CNN”). The output of training was set of parameter weights for the CNN. The CNN was later in-instanced on the Pi and parameter weights applied. ThePi was then able to use the CNN to convert realtime image from the camera to generate left/right control signals to drive the car.



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.

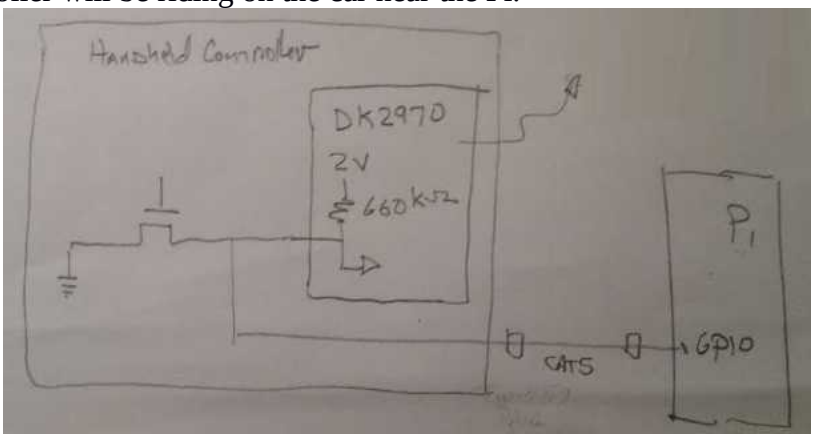
Hacking the Handheld Controller

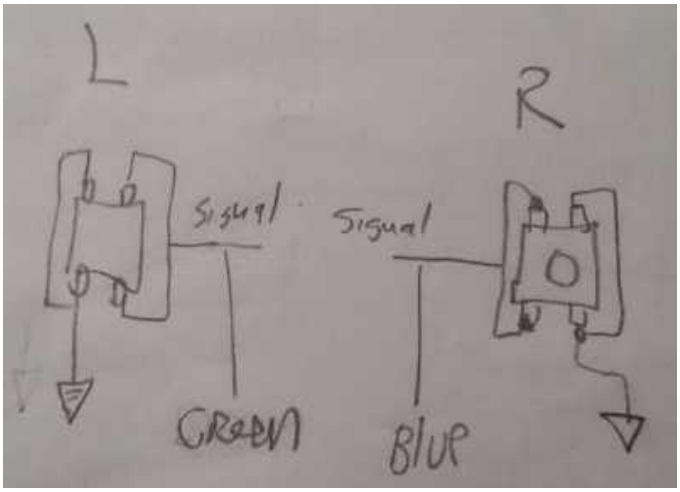
Inside the controller is a small circuit board with four button switches (left/right/forward/reverse), a transmitter chip (DK2970), and analog components to support the antenna. The chip handles

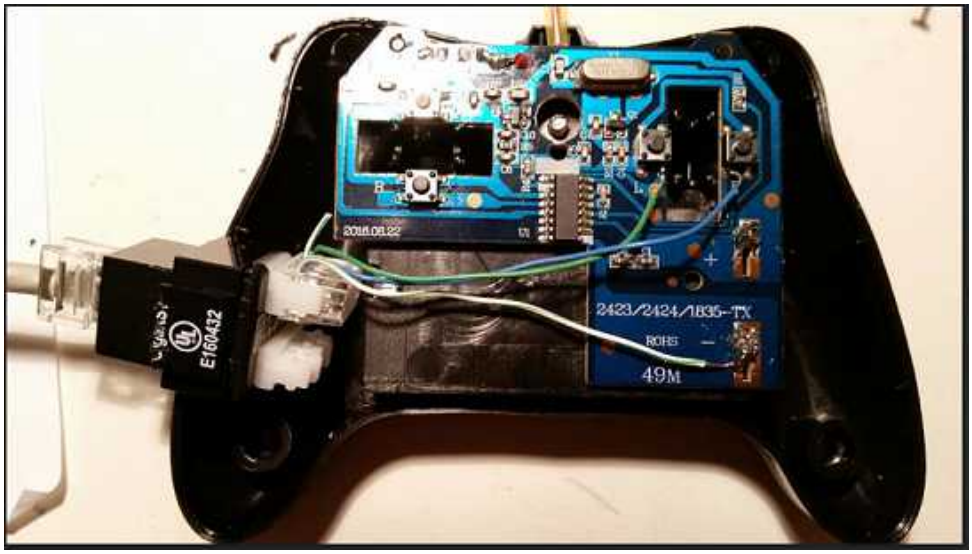
everything - taking in the 4 control signals and outputting the 49mnz carrier. The control signal inputs are 2.1 volt “open drain” - a push of the button grounds the signal. The pullup is internal to the chip and measured at 660kohm. We are fortunate in that signal levels are compatible with the Pi GPIO.

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 the correct approach, it would require 2 wires (a 'receive' and transmit') to/from the Pi for Left and 2 for Right (4 total).
A simpler approach can be pursued given 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.





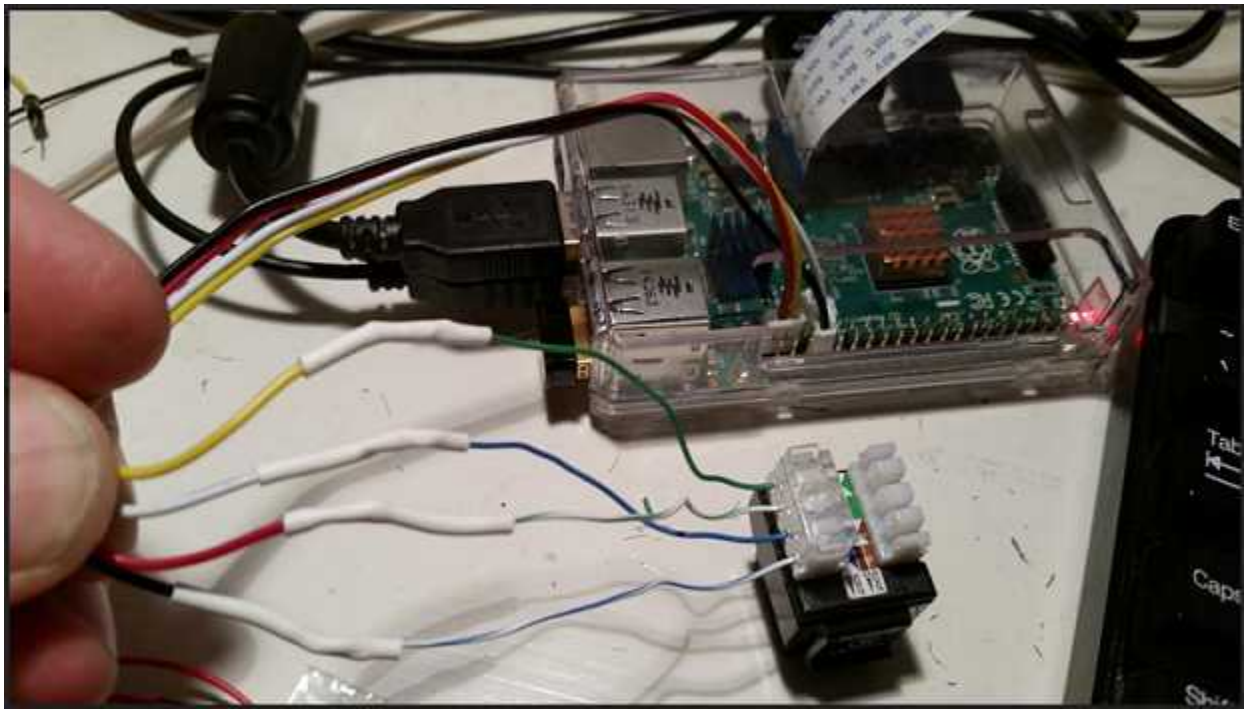

E160432
2423/2424/1835-TX
ROHS
49M

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 are able to demonstrate the Pi can capture the necessary signal and camera training data, and can drive the left/right signals to turn the car. Now it is time to capture data!



The 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 as is available on the Github.

The Dataset

Dataset Overview

The Dataset is a collection of samples taken from a toy car as it is driven around a track. Each sample includes an image from a ftont-facing camera and a steering value which is the direction the car is being steered at that time.

A piece of red tape which identifies the centerline of the track. The background is a white carpet.

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

Images are 90h by 180w using RGB encoding. This is kept as a 4-dimensional numpy array of numpy.uint8. The dimensions of this array are [N][64][64][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 dataset consists of images and corresponding steering values from a RC car using a front-mounted Raspberry PI camera.

Genostrat TrainingData 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 (16,200) 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 was 18922.

Converting Class Variable

Although the Geneostrat car uses the same Raspberry PI as our project, there are minor differences in the hardware associated with the car. The Geneostrat car uses numeric steering values while our car uses categorical class. We thus need to convert the class variable in in the Raw Data from numeric to a categorical class. 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

In other words - there are 20% more CCW (left turning) samples than CW (right turning). The mean of the left hand turns was 50 points off the centerpoint of 400, while the mean of the right hand turns was 30 points off the centerpoint. 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

In other words - the “straight” is closely centered around the value of 400. There are significantly less “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

In other words - the “wandering oval” is a broader distribution than even the R18-25, and slightly favored left hand turns.

Based on the investigation above 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)
- shuffling
- segmenting into training/testing subsets
- saving to pickle file

The CNN

We base our CNN structure on the MNIST example from Keras:
https://github.com/keras-team/keras/tree/master/examples

This is a good starting point because MNIST dataset is similar to ours. Difference are the number of bits in the image and RGB vs black/white image content.

	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	90x180 RGB

The MNIST example CNN is available at
https://github.com/keras-team/keras/blob/master/examples/mnist_mlp.py

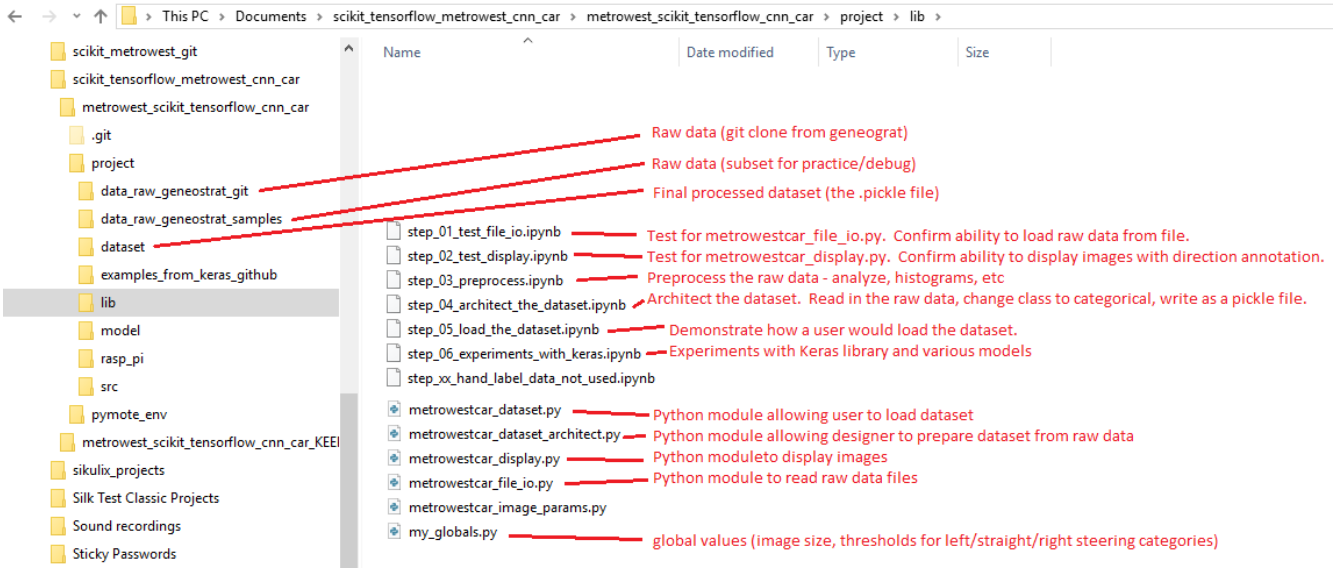
Our CNN is structured as follows:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 90, 180, 32)       896
_________________________________________________________________
activation_1 (Activation)    (None, 90, 180, 32)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 88, 178, 32)       9248
_________________________________________________________________
activation_2 (Activation)    (None, 88, 178, 32)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 44, 89, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 44, 89, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 44, 89, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 44, 89, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 42, 87, 64)        36928
_________________________________________________________________
activation_4 (Activation)    (None, 42, 87, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 21, 43, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 21, 43, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 57792)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               29590016
_________________________________________________________________
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: 29,657,636
Trainable params: 29,657,636
Non-trainable params: 0
_________________________________________________________________
None
```

Dataset Design and Use. Experiments with Keras.

Python code in lib/ provides:

- architect the ability to prepare the dataset
- user the ability to load the dataset
- experiments using the Keras library



Appendix 1:
The following code was used when hacking the Handheld Contrlller 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)
```