Self-driving Car Steering Wheel Angle Predictions

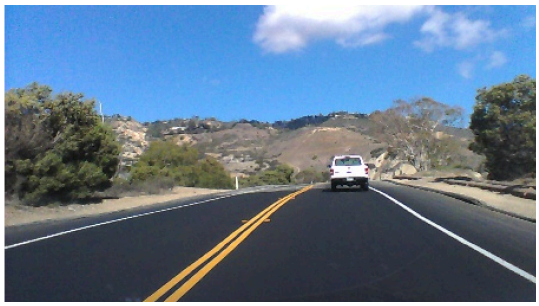
Emma L
JAN 11TH,2020

GOAL OF THE PROJECT

- To predict steering wheel angles based on road images.

DATA SOURCE:

- Dataset (dataset 1) URL: https://github.com/SullyChen/driving-datasets
- The dataset contains approximately 45,500 images, 2.2GB. Data was recorded around Rancho Palos Verdes and San Pedro California in 2017.
- The data format is as follows: filename.jpg angle
- Examples of images:



STEPS:

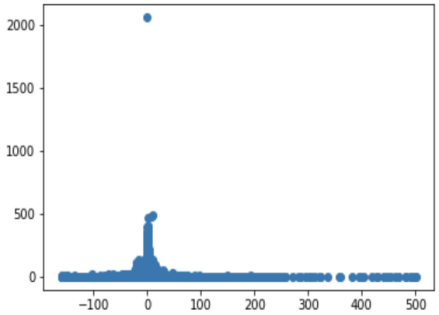
1. **Download Dataset**
2. **Data Preprocessing**
    - Round angles
    - Drop data points whose angle occurs less than 200 times
    - Randomly drop elements whose angle occurs more than 3,000 times so that the maximum number of occurrences of each angle is 3,000
    - Remove outliers
    - Scale pixel data to a 0 - 1 range
3. **Design and Implement CNN Model**
    - Design a CNN model
    - Split the dataset into train, validation, and test data
    - Create a batch generator
    - Implement the model
    - Test the model
4. **Display and Analyze Results**

Step 1: Download Dataset

Step 2: Data Preprocessing

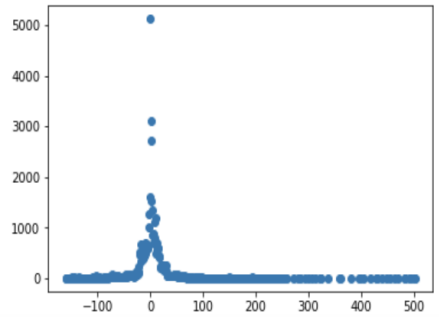
As can be seen from the graph below, the distribution of angles does not follow a normal distribution, so we can process data to make the distribution closer to a normal distribution.

```
 0.00      2061
 9.78       485
 8.27       477
 1.51       467
 1.31       406
           ...
 80.87        1
 27.92        1
 70.99        1
-66.25        1
 126.25       1
Name: angles, Length: 2377, dtype: int64
```

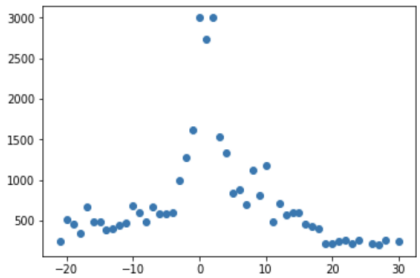


We can also find that angles are rounded to two decimal places. However, for example, there is not much difference between 1.51 degrees and 1.31 degrees, so we can round the numbers to the nearest integer. After rounding the angles, I got a distribution of angles as shown below:

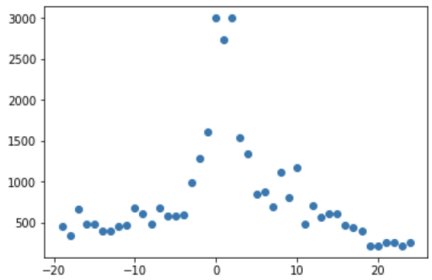
```
 0.0       5122
 2.0       3119
 1.0       2730
-1.0       1616
 3.0       1535
           ...
 359.0        1
 285.0        1
 158.0        1
 234.0        1
 112.0        1
Name: angles, Length: 430, dtype: int64
```



Angles in certain ranges do not occur often. It would be difficult to predict in those ranges, so we can drop data those data points. In addition, some angles occur much more often than others, making the dataset imbalanced. To address this, I randomly dropped data points whose angle occurs more than 3,000 times so that the maximum number of occurrences of each angle is 3,000. After dropping some data points, the distribution is closer to normal distribution. The next step is to remove outliers. Below are distributions before and after removing outliers.



Before removing outliers

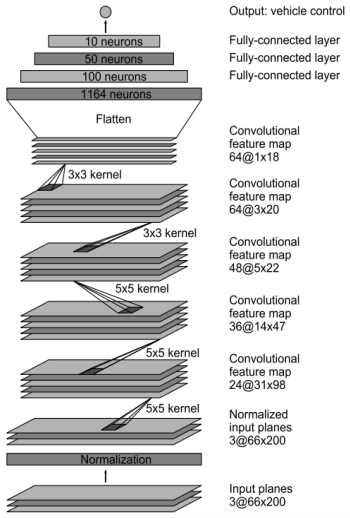


After removing outliers

For Deep Learning Neural Network models, the scale of data matters. Therefore, before implementing a CNN model, we also need to scale pixel data to a 0 - 1 range.

Step 3: Design and Implement Model

Among all the architectures I have tried, the one used in a NVIDIA paper creates the best result:



I split the dataset into training, validation, and test data with each accounts for 75%, 22%, and 3% of the whole dataset. They have 26,340, 7,726, and 1,053 data points respectively.

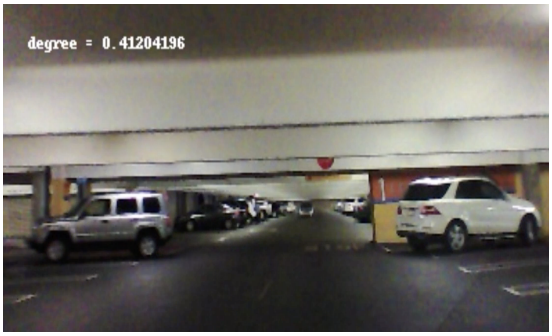
Since the dataset contains a large number of data points, I created a batch generator to train the model on training and validation data.

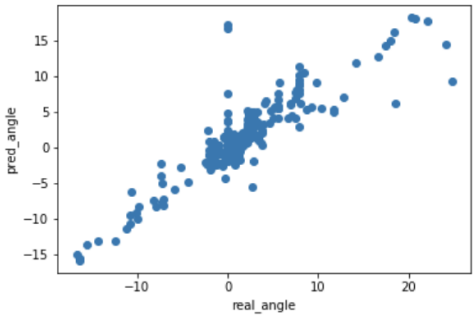
```
Epoch 1/30
411/411 [==============================] - 135s 327ms/step - loss: 45.9448 - val_loss: 27.7976
Epoch 2/30
411/411 [==============================] - 128s 312ms/step - loss: 18.3505 - val_loss: 16.1391
Epoch 3/30
411/411 [==============================] - 128s 312ms/step - loss: 10.9779 - val_loss: 11.5818
Epoch 4/30
411/411 [==============================] - 126s 307ms/step - loss: 7.5080 - val_loss: 8.4311
Epoch 5/30
411/411 [==============================] - 130s 317ms/step - loss: 5.7725 - val_loss: 7.2556
Epoch 6/30
411/411 [==============================] - 135s 328ms/step - loss: 4.8474 - val_loss: 6.6091
Epoch 7/30
411/411 [==============================] - 129s 314ms/step - loss: 3.8900 - val_loss: 5.9368
Epoch 8/30
411/411 [==============================] - 126s 306ms/step - loss: 3.2140 - val_loss: 5.4120
Epoch 9/30
411/411 [==============================] - 123s 300ms/step - loss: 3.1593 - val_loss: 4.7094
Epoch 10/30
411/411 [==============================] - 125s 303ms/step - loss: 2.8448 - val_loss: 4.8982
Epoch 00010: early stopping
```

After running the test data against the model, I got a mean squared error of 5.244 and R^2 of 0.941. The graph below shows the result.

Step 4: Display and Analyze Results

I created a video from the first 250 images to display the result. The video URL is:
https://github.com/emmayylu/Self-Driving-Car-Angle-Prediction/blob/master/result.mov





As can be seen from the graph on the right, some results are vastly different from the actual values. The table below contains images whose predicted values are larger than 15 degrees (the angles in the table are the actual values). The inaccuracies in the predictions on `126.jpg` and `130.jpg` are caused by sudden changes in the steering wheel angles. There is an oncoming car in both images, so the steering wheel should be pushed in a clockwise motion. Taking `126.jpg` as an example, even though `125.jpg` and `126.jpg` have almost the same images, the actual angles are vastly different (18.45 and 0), and the model has learned that the car should steer to the right and, therefore, generate predict the angles for the two images to be 16.111744 and 17.206934.

	files	angles
124	124.jpg	18.05
125	125.jpg	18.45
126	126.jpg	0.00
127	127.jpg	20.27
128	128.jpg	20.77
129	129.jpg	22.08
130	130.jpg	0.00