REPORT AND CODE FOR THE BEHAVIORAL CLONING PROJECT UDACITY

Create a Training Set

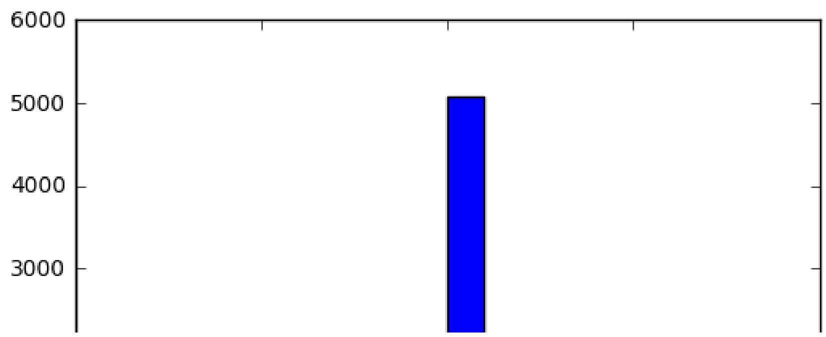
The first problem incontered during this project has been the record of data in order to train our CNN. The creation of a precise and extended dataset has revailed to be one of the key point of this project. There are two main issues incontered in this phase:

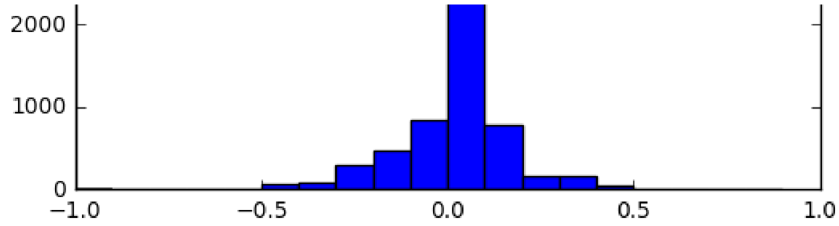
```
- The simulator is hard to use with just a keyboard
- The first track presents many left turns and not enough right turns
```

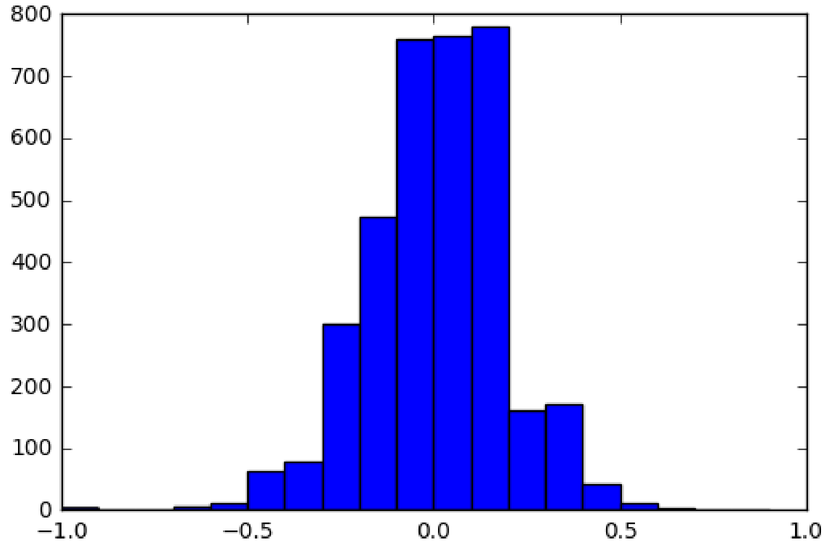
In order to solve this two problem the car has been moved using a mouse controller and the data has been recorded driving the car along the track in clockwise and anticlockwise. After many attempts the best Dataset reveals to be in any case the one released by Udacity, that has been for train this CNN. The Udacity dataset presented some problem infact the class of angle from -0.1 to 0.1 has been overrapresented compared to the others, for solve this problem 4400 samples with angles in this range has been removed from the dataset. The two histogram below here presents the sample distribution before and after the filtering.

In [9]:

```
import os
import csv
import math
import numpy as np
import matplotlib.pyplot as plt
import sklearn

PathToDatasetCSV='TEST/data/data/driving_log.csv'
samples = []
with open(PathToDatasetCSV) as csvfile:
    reader = csv.reader(csvfile)
    for line in reader:
        samples.append(line)


sklearn.utils.shuffle(samples)
angle_list=[float(row[3]) for row in samples]
bin=np.arange(-1,1,0.1)
plt.hist(angle_list,bins=bin)
plt.show()

remove_list = []
remove_element_counter=0
for i in range(len(angle_list)):
    if(angle_list[i]>=-0.01 and angle_list[i]<=0.01):
        remove_list.append(i)
        remove_element_counter=remove_element_counter+1
    if(remove_element_counter>=4400):
        break

samples = np.delete(samples, remove_list,axis=0)
angle_list = np.delete(angle_list, remove_list)

bin=np.arange(-1,1,0.1)
plt.hist(angle_list,bins=bin)
plt.show()
```







Preprocess the images:

The images have been preprocessed in order to reveal more feature and reduce ambient light problem:

```
-The image have been cropped in order to remove peaces of the image out of the road

-A Gaussian filter has been applyed in order to remove the noise in the image and make it smooth
er

-The image have been resized in order to fit it with the NVIDIA end to end CNN to 200*66*3

-The contrast of the image have been increased
```

In [10]:

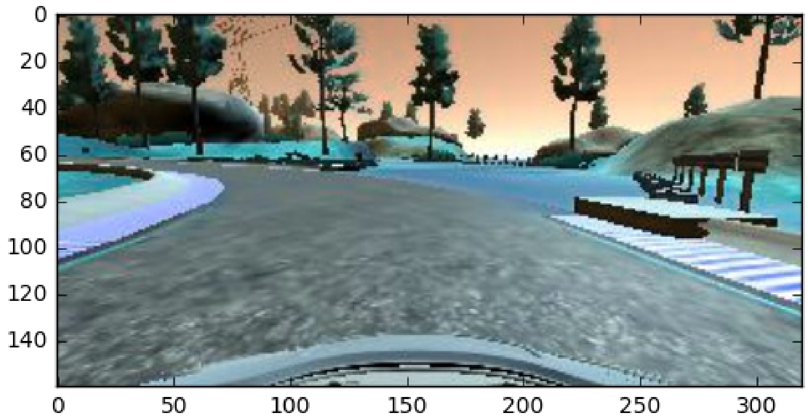
```python
import matplotlib.pyplot as plt
import scipy.misc
import csv
import math
import cv2
with open('TEST/data/data/driving_log.csv', 'r') as csvfile:
    spamreader = csv.reader(csvfile)
    label=[]
    image=[]
    for row in spamreader:
        name_center_image = './TEST/data/data/IMG/'+row[0].split('/')[-1]
        center_image = cv2.imread(name_center_image)
        image.append(center_image)
        steering_center = float(row[3])
        steering_center_flipped = -steering_center
        #create adjusted steering measurements for the side camera images
        correction = 0.25 # this is a parameter to tune
        steering_left = float(steering_center + correction)
        steering_left_flipped = - steering_left
        steering_right = float(steering_center - correction)
        steering_right_flipped = - steering_right
        label.append(steering_center)
        #label.append(steering_center_flipped)
        #label.append(steering_left)
        #label.append(steering_left_flipped)
        #label.append(steering_right)
        #label.append(steering_right_flipped)
```

```
        #label.append(steering_right_flipped)
```

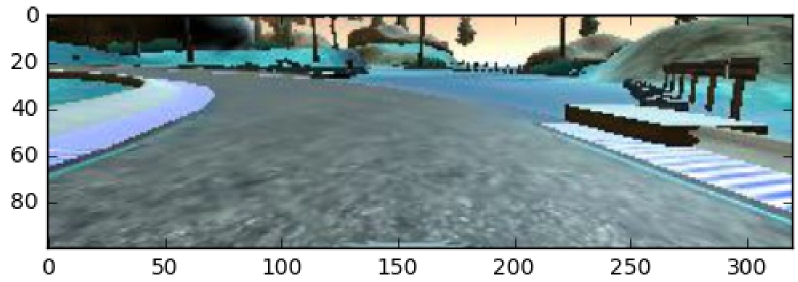
In [11]:

```
from matplotlib import pyplot as plt
import cv2
img=image[5]
print("ORIGINAL:")
print(img.shape)
plt.imshow(img)
plt.show()
x,y,z = image[0].shape
crop_img = img[40:140, 0:y]
print("IMAGE CROPPED:")
print(crop_img.shape)
plt.imshow(crop_img)
plt.show()
crop_img = cv2.GaussianBlur(crop_img, (3,3),0)
print("GAUSSIAN FILTERED IMAGE:")
print(crop_img.shape)
plt.imshow(crop_img)
plt.show()
crop_img = cv2.resize(crop_img, (200,66))
print("RESIZED IMAGE:")
print(crop_img.shape)
plt.imshow(crop_img)
plt.show()
crop_img = cv2.cvtColor(crop_img, cv2.COLOR_BGR2HSV)
print("HIGH CONTRAST IMAGE:")
print(crop_img.shape)
plt.imshow(crop_img)
plt.show()
```

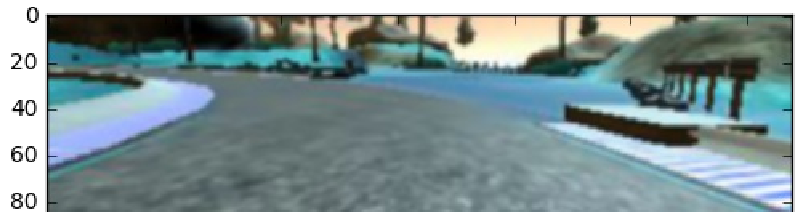
```
ORIGINAL:
(160, 320, 3)
```



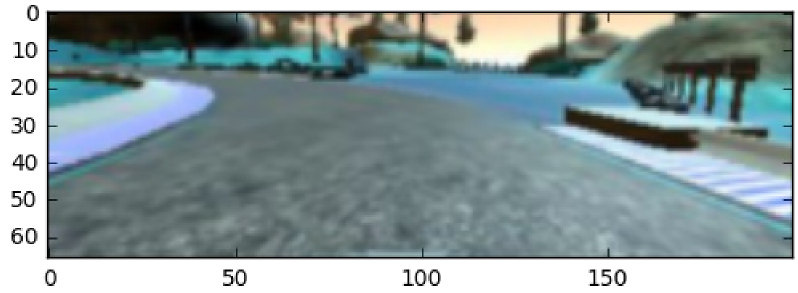

```
IMAGE CROPPED:
(100, 320, 3)
```



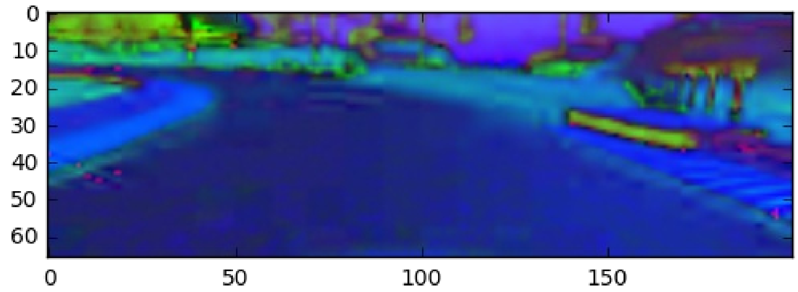

```
GAUSSIAN FILTERED IMAGE:
(100, 320, 3)
```




```
RESIZED IMAGE:
(66, 200, 3)
```




```
HIGH CONTRAST IMAGE:
(66, 200, 3)
```



CNN Architecture and Generator

The dataset revealed to be very big and hard to handle the creation by the pc. In order to solve this problem a generator has been used. Moreover inside the generator the preprocess image have been used and the image from the left and right camera in order to increase the dataset size. The dataset has been increased first flipping the image and inverting the angle associated (- angle) and second using the left and right images introducing a correction due to the position of the cameras of 0.25 (left_camera_angle=center_camera_angle + 0.25 , right_camera_angle=center_camera_angle - 0.25).

The CNN Architecture used in this project is the one presented in NVIDIA's End to End Learning for Self-Driving Cars paper resumed down here:

==

lambda_1 (Lambda) (None, 66, 200, 3) 0 lambda_input_1[0][0]

---

convolution2d_1 (Convolution2D) (None, 31, 98, 24) 1824 lambda_1[0][0]

---

dropout_1 (Dropout) (None, 31, 98, 24) 0 convolution2d_1[0][0]

---

convolution2d_2 (Convolution2D) (None, 14, 47, 36) 21636 dropout_1[0][0]

---

dropout_2 (Dropout) (None, 14, 47, 36) 0 convolution2d_2[0][0]

---

convolution2d_3 (Convolution2D) (None, 5, 22, 48) 43248 dropout_2[0][0]

---

dropout_3 (Dropout) (None, 5, 22, 48) 0 convolution2d_3[0][0]

---

convolution2d_4 (Convolution2D) (None, 3, 20, 64) 27712 dropout_3[0][0]

---

dropout_4 (Dropout) (None, 3, 20, 64) 0 convolution2d_4[0][0]

---

convolution2d_5 (Convolution2D) (None, 1, 18, 64) 36928 dropout_4[0][0]

---

flatten_1 (Flatten) (None, 1152) 0 convolution2d_5[0][0]

---

dense_1 (Dense) (None, 1164) 1342092 flatten_1[0][0]

---

dense_2 (Dense) (None, 100) 116500 dense_1[0][0]

---

dense_3 (Dense) (None, 50) 5050 dense_2[0][0]

---

dense_4 (Dense) (None, 10) 510 dense_3[0][0]

---

dense_5 (Dense) (None, 1) 11 dense_4[0][0]
==

This CNN is the same one presented in the paper I just introduced a Dropout with a probability of 0.2 in this model and the activation choosen was a relu and a tanh for the last layer.

The dataset has been divided using the 80% of the data for the training set and the 20% for the validation set. The CNN has been compiled using the ADAM optimized and the mean square error. The mean square error(MSE) reveals to not be a good indicator of the quality of the net in fact a small MSE does not indicate alwais a better performing in simulator net.

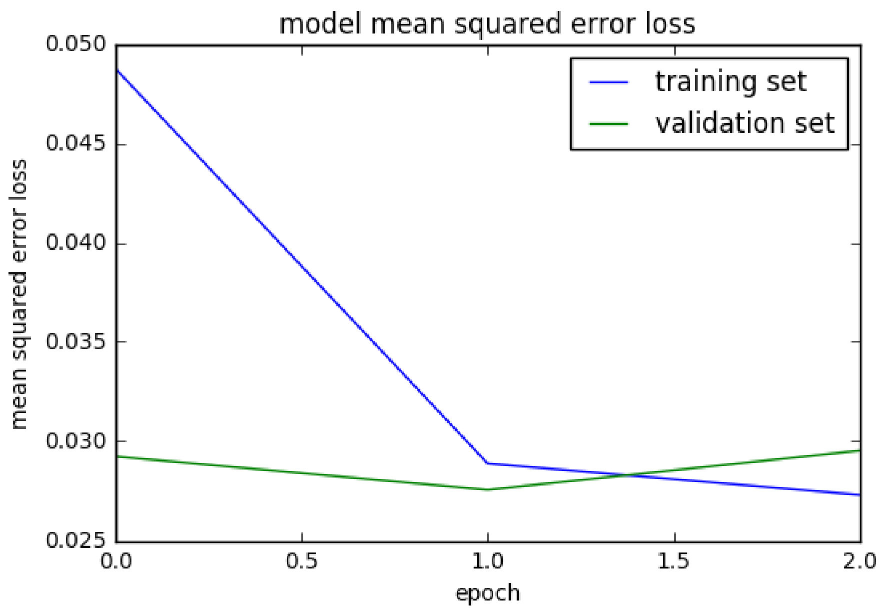
In [18]:

```
from sklearn.model_selection import train_test_split
train_samples, validation_samples = train_test_split(samples, test_size=0.2)

import cv2
import numpy as np
import sklearn


def preprocess_image(image):
    x,y,z=image.shape
    image = image[40:140, 0:y]
    image = cv2.GaussianBlur(image, (3,3),0)
    image = cv2.resize(image, (200,66))
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    return image

def generator(samples, batch_size=100):
    num_samples = len(samples)
    while 1: # Loop forever so the generator never terminates
        sklearn.utils.shuffle(samples)
        for offset in range(0, num_samples, batch_size):
            batch_samples = samples[offset:offset+batch_size]

            images = []
            angles = []
            for batch_sample in batch_samples:
                name_center_image = './TEST/data/data/IMG/'+batch_sample[0].split('/')[-1]
                name_left_image = './TEST/data/data/IMG/'+batch_sample[1].split('/')[-1]
                name_right_image = './TEST/data/data/IMG/'+batch_sample[2].split('/')[-1]
                center_image = cv2.imread(name_center_image)
                center_image_flipped= np.fliplr(center_image)
                left_image = cv2.imread(name_left_image)
                left_image_flipped= np.fliplr(left_image)
                right_image = cv2.imread(name_right_image)
                right_image_flipped= np.fliplr(right_image)
                center_angle =float(batch_sample[3]) #math.ceil(float(batch_sample[3])*100)/100
                center_angle_flipped = - center_angle
                correction = 0.25
                left_angle = center_angle + correction
                left_angle_flipped = - left_angle
                right_angle = center_angle - correction
                right_angle_flipped = - right_angle
                images.append(preprocess_image(center_image))
                images.append(preprocess_image(center_image_flipped))
                images.append(preprocess_image(left_image))
                images.append(preprocess_image(left_image_flipped))
                images.append(preprocess_image(right_image))
                images.append(preprocess_image(right_image_flipped))
                angles.append(center_angle)
                angles.append(center_angle_flipped)
                angles.append(left_angle)
                angles.append(left_angle_flipped)
                angles.append(right_angle)
                angles.append(right_angle_flipped)

            # trim image to only see section with road
```

```
            X_train = np.array(images)
            y_train = np.array(angles)
            yield sklearn.utils.shuffle(X_train, y_train)

# compile and train the model using the generator function
train_generator = generator(train_samples, batch_size=100)
validation_generator = generator(validation_samples, batch_size=100)

ch, row, col = 3, 66,200#3, 160, 320  # Trimmed image format

from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, BatchNormalization
from keras.layers import Convolution2D, MaxPooling2D, Lambda, Cropping2D

model = Sequential()
model.add(Lambda(lambda x: x/127.5 - 1.,input_shape=(row,col,ch)))
model.add(Convolution2D(24,5,5,border_mode='valid', activation='relu', subsample=(2,2)))
model.add(Dropout(0.2))
model.add(Convolution2D(36,5,5,border_mode='valid', activation='relu', subsample=(2,2)))
model.add(Dropout(0.2))
model.add(Convolution2D(48,5,5,border_mode='valid', activation='relu', subsample=(2,2)))
model.add(Dropout(0.2))
model.add(Convolution2D(64,3,3,border_mode='valid', activation='relu', subsample=(1,1)))
model.add(Dropout(0.2))
model.add(Convolution2D(64,3,3,border_mode='valid', activation='relu', subsample=(1,1)))
model.add(Flatten())
model.add(Dense(1164, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='tanh'))

model.summary()

model.compile(loss='mse', optimizer='adam')
history_object = model.fit_generator(train_generator, samples_per_epoch=(len(train_samples)*6), validat
ion_data=validation_generator, nb_val_samples=len(validation_samples), nb_epoch=3)

print(history_object.history.keys())

### plot the training and validation loss for each epoch
plt.plot(history_object.history['loss'])
plt.plot(history_object.history['val_loss'])
plt.title('model mean squared error loss')
plt.ylabel('mean squared error loss')
plt.xlabel('epoch')
plt.legend(['training set', 'validation set'], loc='upper right')
plt.show()



model.save("model.h5")
```

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
lambda_7 (Lambda)                (None, 66, 200, 3)    0           lambda_input_7[0][0]
____________________________________________________________________________________________________
convolution2d_31 (Convolution2D) (None, 31, 98, 24)    1824        lambda_7[0][0]
____________________________________________________________________________________________________
dropout_25 (Dropout)             (None, 31, 98, 24)    0           convolution2d_31[0][0]
____________________________________________________________________________________________________
convolution2d_32 (Convolution2D) (None, 14, 47, 36)    21636       dropout_25[0][0]
____________________________________________________________________________________________________
dropout_26 (Dropout)             (None, 14, 47, 36)    0           convolution2d_32[0][0]
____________________________________________________________________________________________________
convolution2d_33 (Convolution2D) (None, 5, 22, 48)     43248       dropout_26[0][0]
____________________________________________________________________________________________________
dropout_27 (Dropout)             (None, 5, 22, 48)     0           convolution2d_33[0][0]
____________________________________________________________________________________________________
convolution2d_34 (Convolution2D) (None, 3, 20, 64)     27712       dropout_27[0][0]
____________________________________________________________________________________________________
dropout_28 (Dropout)             (None, 3, 20, 64)     0           convolution2d_34[0][0]
____________________________________________________________________________________________________
convolution2d_35 (Convolution2D) (None, 1, 18, 64)     36928       dropout_28[0][0]
____________________________________________________________________________________________________
flatten_7 (Flatten)              (None, 1152)          0           convolution2d_35[0][0]
```

```
dense_31 (Dense)                 (None, 1164)          1342092     flatten_7[0][0]
____________________________________________________________________________________________________
dense_32 (Dense)                 (None, 100)           116500      dense_31[0][0]
____________________________________________________________________________________________________
dense_33 (Dense)                 (None, 50)            5050        dense_32[0][0]
____________________________________________________________________________________________________
dense_34 (Dense)                 (None, 10)            510         dense_33[0][0]
____________________________________________________________________________________________________
dense_35 (Dense)                 (None, 1)             11          dense_34[0][0]
====================================================================================================
Total params: 1,595,511
Trainable params: 1,595,511
Non-trainable params: 0
____________________________________________________________________________________________________
Epoch 1/3
17448/17448 [==============================] - 19s - loss: 0.0488 - val_loss: 0.0292
Epoch 2/3
17448/17448 [==============================] - 18s - loss: 0.0289 - val_loss: 0.0276
Epoch 3/3
17448/17448 [==============================] - 17s - loss: 0.0273 - val_loss: 0.0295
dict_keys(['loss', 'val_loss'])
```



Conclusion

In conclusion this has been a real challenging project. This project has revealed to me the importance of a good dataset and preprocess procedure, surprisingly a good filtering of the dataset has revealed more improuvments than modifying the net.

Future Work

There are many areas to be explored in order to improve my work.

```
-First a better training set and a better filtering of the training set I'm sure would improve t
he model

-Second exploring a more sofisticate procedure for my images and moreover explore other way to i
ncrease the dataset such as distort images.
```

In [ ]: