In [1]:

```
# Loading the packages
import pandas
import cv2
import numpy
import matplotlib.pyplot as plt
```

In [2]:

```
# loading the csv file using pandas.
df = pandas.read_csv(r"data/driving_log.csv",header=None)
# checking the first five rows
df.head()
```

Out[2]:

	0	1	2	3	4	5	
0	D:\AI\nanodegree\Project - Behavioral clonning...	D:\AI\nanodegree\Project - Behavioral clonning...	D:\AI\nanodegree\Project - Behavioral clonning...	0.0	0.0	0	0.71
1	D:\AI\nanodegree\Project - Behavioral clonning...	D:\AI\nanodegree\Project - Behavioral clonning...	D:\AI\nanodegree\Project - Behavioral clonning...	0.0	0.0	0	0.71
2	D:\AI\nanodegree\Project - Behavioral clonning...	D:\AI\nanodegree\Project - Behavioral clonning...	D:\AI\nanodegree\Project - Behavioral clonning...	0.0	0.0	0	0.70
3	D:\AI\nanodegree\Project - Behavioral clonning...	D:\AI\nanodegree\Project - Behavioral clonning...	D:\AI\nanodegree\Project - Behavioral clonning...	0.0	0.0	0	0.70
4	D:\AI\nanodegree\Project - Behavioral clonning...	D:\AI\nanodegree\Project - Behavioral clonning...	D:\AI\nanodegree\Project - Behavioral clonning...	0.0	0.0	0	0.69

In [3]:

```
#checking the shape of data
df.shape
```

Out[3]:

```
(3400, 7)
```

In [4]:

```
# creating list for storing the images and the respective measurements
images = []
measurements = []
# iterating through the dataframe to read every image one by one
for i in range(df.shape[0]):
    #extracting the steering angle value from 3rd column
    steering_center = df[3][i]
    # creating a correction factor for left and right images
    correction = 0.3 # this parameter is tuned 5 times to get this value
    # using the correction factor getting the measurement for left and right images
    steering_left = steering_center + correction
    steering_right = steering_center - correction
    # image path for center image
    center_path = df[0][i]
    #loading image using opencv
    center_img = cv2.imread(center_path)
    # converting the image from BGR to the RGB format
    center_img = cv2.cvtColor(center_img,cv2.COLOR_BGR2RGB)
    # image path for the left image
    left_path = df[1][i]
    # loading the image using opencv
    left_img = cv2.imread(left_path)
    # converting the image from BGR to RGB using opencv
    left_img = cv2.cvtColor(left_img,cv2.COLOR_BGR2RGB)
    # image path for the right image
    right_path = df[2][i]
    #loading the image
    right_img = cv2.imread(right_path)
    # converting into the RGB format from the BGR format
    right_img = cv2.cvtColor(right_img,cv2.COLOR_BGR2RGB)
    # appending the images
    images.extend((center_img,left_img,right_img))
    # appending the respective measurements
    measurements.extend((steering_center,steering_left,steering_right))
```

In [5]:

```
#checking the total number of images
len(images)
```

Out[5]:

```
10200
```

In [6]:

```
# image augementation using flipping the images and reversing the sign of the respectiv
e measurement
aug_images,aug_measurements = [], []
for i in range(len(images)):
    aug_images.append(images[i])
    aug_measurements.append(measurements[i])
    aug_images.append(cv2.flip(images[i],1))
    aug_measurements.append(measurements[i]*-1.0)
```

In [7]:

```
# converting the images and measurements to numpy array format
xtrain = numpy.array(aug_images)
ytrain = numpy.array(aug_measurements)
```

In [8]:

```
print(xtrain.shape)
print(ytrain.shape)
```

```
(20400, 160, 320, 3)
(20400,)
```

In [9]:

```
from keras import models,layers
```

```
Using TensorFlow backend.
```

In [10]:

```
# creating the mdoel
model = models.Sequential()
# adding the lambda function layer to scale the image pixel values
model.add(layers.Lambda(lambda x:x/255.0 - 0.5,input_shape=(160,320,3)))
#adding the layer to crop the images by 70 from top and 20 from bottom
model.add(layers.Cropping2D(cropping=((70,20), (0,0))))
# adding the 5 convolutional layers
model.add(layers.Conv2D(filters = 20, kernel_size=(5,5),activation='relu',strides=(2, 2
)))
model.add(layers.Conv2D(filters = 40, kernel_size=(5,5),activation='relu',strides=(2, 2
)))
model.add(layers.Conv2D(filters = 50, kernel_size=(5,5),activation='relu',strides=(2, 2
)))
model.add(layers.Conv2D(filters = 60, kernel_size=(3,3),activation='relu'))
model.add(layers.Conv2D(filters = 60, kernel_size=(3,3),activation='relu'))
model.add(layers.Flatten())
# adding the dense layers
model.add(layers.Dense(100,activation='relu'))
# drop out optimization
model.add(layers.Dropout(0.5))
# addint the dense layers
model.add(layers.Dense(50,activation='relu'))
model.add(layers.Dense(10,activation='relu'))
# the output layer
model.add(layers.Dense(1))
```

In [11]:

```
# compile the model, optimizer used - RMSProp, loss = MSE
model.compile(optimizer='RMSProp',loss='mse')
# train the model with 10 epochs and 20% validation split
history_object = model.fit(xtrain,ytrain,
                           validation_split=0.2,
                           shuffle=True,
                           epochs=10,
                           batch_size=128,
                           verbose=True)
```

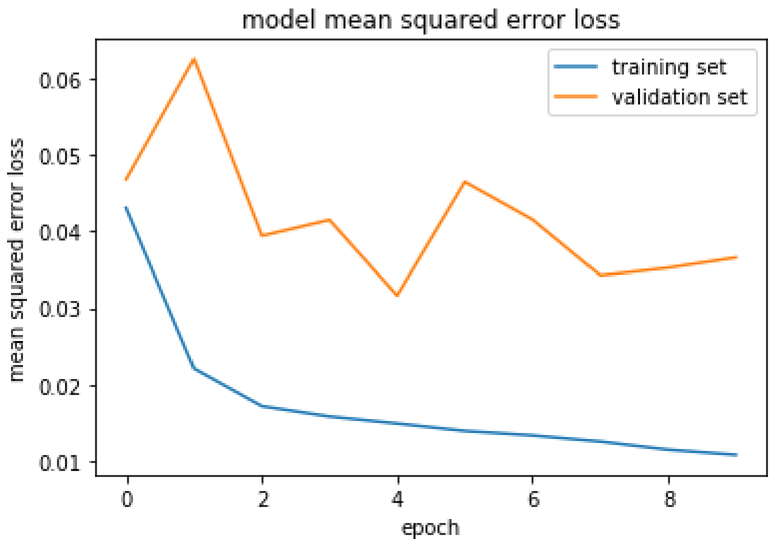
```
Train on 16320 samples, validate on 4080 samples
Epoch 1/10
16320/16320 [==============================] - 48s 3ms/step - loss: 0.0430
- val_loss: 0.0467
Epoch 2/10
16320/16320 [==============================] - 40s 2ms/step - loss: 0.0221
- val_loss: 0.0625
Epoch 3/10
16320/16320 [==============================] - 44s 3ms/step - loss: 0.0172
- val_loss: 0.0394
Epoch 4/10
16320/16320 [==============================] - 43s 3ms/step - loss: 0.0158
- val_loss: 0.0415
Epoch 5/10
16320/16320 [==============================] - 42s 3ms/step - loss: 0.0149
- val_loss: 0.0316
Epoch 6/10
16320/16320 [==============================] - 40s 2ms/step - loss: 0.0139
- val_loss: 0.0464
Epoch 7/10
16320/16320 [==============================] - 42s 3ms/step - loss: 0.0134
- val_loss: 0.0415
Epoch 8/10
16320/16320 [==============================] - 43s 3ms/step - loss: 0.0126
- val_loss: 0.0342
Epoch 9/10
16320/16320 [==============================] - 46s 3ms/step - loss: 0.0115
- val_loss: 0.0352
Epoch 10/10
16320/16320 [==============================] - 43s 3ms/step - loss: 0.0108
- val_loss: 0.0366
```

In [12]:

```
# save the trained model
model.save('model.h5')
```

In [13]:

```
# analysing the loss function for train and validation data
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()
```



In [ ]:

```
```