**SHARVARI (After Arun finishes saying about what has been done till review-2 in short)**

So now to train our model, we need to develop a model architecture first. We’ll use Nvidia’s model which is used in real life self-driving cars as well. It is a deep convolution network that works well for regression problems.

This is Nvidia’s model and it consists of many sequential layers. So we’ll define the model as a sequential one to add on different layers. The first one is an input layer which is given and the next layer is to normalize the input, but since we already did it during our image pre-processing, we skip it here. And the next is a convolution layer with 24 filters having a kernel of 5x5. We add that layer to our sequential model by passing these to the Convolution 2D function along with the stride length of (2,2) which denotes that we move 2 pixels horizontally and 2 pixels vertically at a time. The next argument is the input shape, that is the dimensions which is 66x200x3 and we use the relu activation function. The next 4 layers are also convolutional layers and they are similarly added to our model with 36,48,64,64 filters and kernel size of 5x5 and 3x3. And the next is a flatten layer which converts the output from the last convolution layer to a one dimensional array, here 64x18 is converted to a 1-D array of size 1152 neurons and passed to the fully connected layers. These dense layers reduce the number of neurons after each step, first it’s made to 100 neurons, and then 50 and then 10 and at last we are returned 1 neuron, which denotes the particular steering angle value returned.

In between dropout layers are added prevents overfitting of the model. Here dropout(0.5) signifies that 50% of the input values are randomly set to zero.

**RUTHVIK (After Sharvari finishes saying abot the layers and adding them to our model)**

We compile this model by using minimum square error metric and Adam optimizer.

We then print the summary of the model, and the summary shows 5 convolution layers with their respective filters and shape size, and then the dropout, flatten and 3 dense layers.

Now our model is ready to be trained and we train it with our training data for 10 epochs with 300 steps per epoch for our training data and 200 steps for our validation data.

The output of this cell here shows the time taken and the training and validation loss for each epoch and we’ve plotted the loss values and we could see that the validation loss is lesser than the training loss, thus it has a bit of underfitting which is good for our model. We then download and save our model in a folder named Behavioral Cloning.

**ARUN**

Now that we’ve created our trained model that would predict a steering angle value for a particular road image, we need to establish a connection between the driving simulator and the model for them to communicate. So we’ll use socket.io a javascript library for realtime web applications that enables bi-directional communication between the client and the server. We need to save the socket.io code in the same folder as the model being saved.

**ADITYA (After Arun finishes about explaining the need for connection to be established and after the code is opened in Atom)**

We import the required libraries and load our saved model, model.h5 in line 51 and as soon as a connection is established it executes the connect event handler in line 38 and prints connected in our terminal and sends steering angle value 0 and throttle value 0 to the send control function in line 43 which emits these values to the simulator, thus denoting that initially the car is stationary facing the forward direction. Now when the car starts to move it sends the road image in front of it and the telemetry event handler in line 24 is invoked. The image is pre-processed and now sent to the model and the model predicts the steering angle for that image and passes the steering angle value and the throttle value to the send control function which emits these values to the simulator thus helping our car steer to that particular angle and drive. The throttle value passed is such that speed is always less than the speed limit value of 10 set here. So at every instance of our path, the simulator sends the road image in front of it to the model via socket.io and the model predicts the steering angle for that image and send the value to the simulator via the socket.io thus helping in our car steering to a particular angle and driving on its own.

**ARUN**

Will run the code and will show sir the working and explain it

**RUTHVIK (After Arun says about the output working)**

Explain the outcomes slide (slide 21)

**ADITYA (After Ruthvik says about the outcomes)**

Explain that we’ve taken a video of our simulation and have uploaded in google drive and pasted the link here (slide 22)

**SHARVARI (After Aditya says about the video link)**

Explain the impact to society slide (slide 23) and conclude