1. Here we have cloned our Github repository to access the data obtained by driving it in the simulator and
2. Displayed what files are there in the track repository, the output shows the IMG folder and the driving log excel file.
3. We install the library imgaug that’ll be used for image augmentation functions later and
4. We’ve imported the required libraries for the code.
5. Now we create a data directory datadir of the track repository and name the columns as shown('center', 'left', 'right', 'steering', 'throttle', 'reverse', 'speed'). We read the csv file and display its contents, only the first five rows of the file are displayed as an illustration. We could see that it has a center image, left image, a right image file path and the required values.
6. The output displayed above had the full pathname of every image which wasn’t required, so we remove that by splitting the pathname to head and tail and just displaying the tail for every image. The first five rows of the file are displayed.
7. Now we plot the steering angle values for every image by dividing the interval [-1,1] to 25 intervals as shown in the output.
8. The intervals are adjusted using an operation to make zero come in the interval because maximum number of steering angles have a value of 0 and the steering angle distribution is plotted.

14, 15. Our dataset now only has some 1000 images which wouldn’t be enough to train the model accurately, so we generate many more images by using different augmentation techniques to enlarge our dataset. First we apply zoom on an image using the affine function in the image augmentation library on a scale from 1 to 1.3, i.e. zooming it by a maximum of 30%. We plot a random original image as well as the zoomed image and we could see that the image on the right side is a bit zoomed.

16, 17. Next we pan the image, i.e., translate the image horizontally either 10% to the left or 10% to the right, similarly vertically between 10% top or 10% bottom. We plot a random original image as well as the panned image and we could see that the image on the right side is a panned either horizontally/vertically.

18, 19. Next augmentation technique is to alter the brightness of the image, we multiply the image by a factor between (0.2,1.2). Multiplying by a factor less than 1 makes the image dimmer and by a factor greater than 1 makes the image a bit brighter. We plot a random original image as well as the brightness altered image and we could see that the image on the right side has its brightness altered.

20, 21. The last technique involves flipping the image. When flipping an image, the steering angle of the new image is also changed and set to the negative of the steering angle of the original image. We plot a random original image as well as its flipped image and we could see that the original image on the left had a steering angle of x and the flipped image on the right side is flipped and has a steering angle of -x.

22. Now we combine the image augmentation techniques explained by my teammate Aditya. Applying every image augmentation technique on every image makes our dataset too complicated and imbalanced. So we apply the four augmentation techniques on 50% of the images randomly using the condition np.random.rand() < 0.5. random.rand() gives a number between 0 and 1 with equal probability and whenever it is less than 0.5, i.e., for a 50% chance we apply the augmentation techniques, i.e., for 50% of the images. This is followed before applying each augmentation technique.

23. We now shows the images generated by applying these techniques below. We have plotted the 10 random original images and their augmented images. For the 1st image we can see that, it’s xxxxxx has been altered (explain whatever operation has been as visible). (Explain the same for 2/3 images)

24. The next important step before passing our dataset images and the steering angles to the convolutional neural network(CNN) is to preprocess these images. We first crop the images so the sky part and the sceneries are removed. From the image plotted we could see that coordinates 0-60 has the sky part and the scenery and above 135 has the car’s part and we’ll require only the part between 60-135, so we’ll crop out the remaining part. Next we convert the RGB image to a YUV format image, even though RGB images are more attractive it’s easier to train the neural networks work with YUV images and then Gaussian blur is applied on the image which makes it more smoother. We then resize the image to 200x66 and the image is normalized (even though we won’t be able to identify it visually).

25. We plot a random original image as well as the pre-processed image applying the above pre-processing techniques.

26.27 Applying all the image augmentation and the pre-processing techniques would result in an n number of images that would enlarge our dataset, but it would require so much memory to store it. Thus we use a batch generator that helps applying the techniques on the fly, i.e., when the images are being sent to the CNN so that these newly created images needn’t be stored separately. We pass both the training set and the validation set to our batch generator and we applying both image augmentation and the pre-processing techniques on the training set but only pre-processing techniques on the validation set. We’ve then plotted a random training set image and a validation set image to illustrate the visual difference.

Expected Outcomes

We’ve used behavioural cloning technique to train our model and we’ll evaluate its performance in autonomous mode in a different track. If we train the car properly with a balanced dataset, it will perform very well in the second track and drive on its own. But if we don’t train the car properly then it may keep skewing to the left or right and may also go off the track