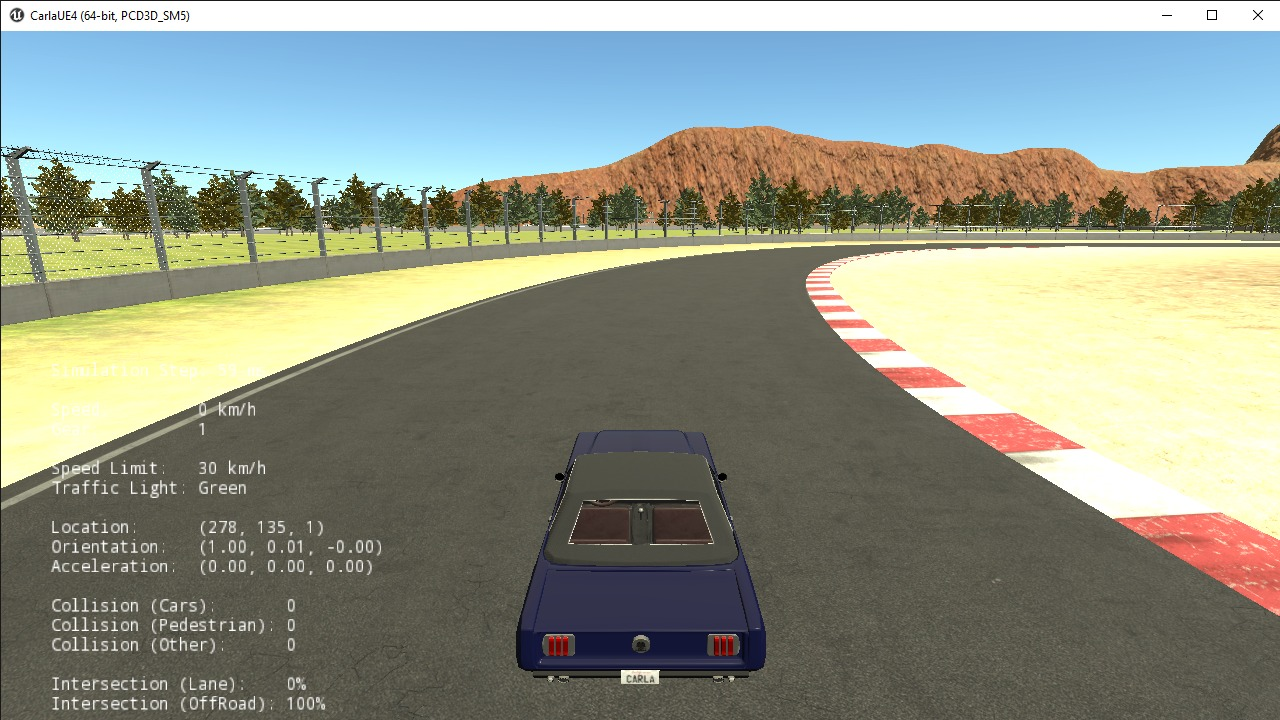
**SELF DRIVING CAR PROJECT**

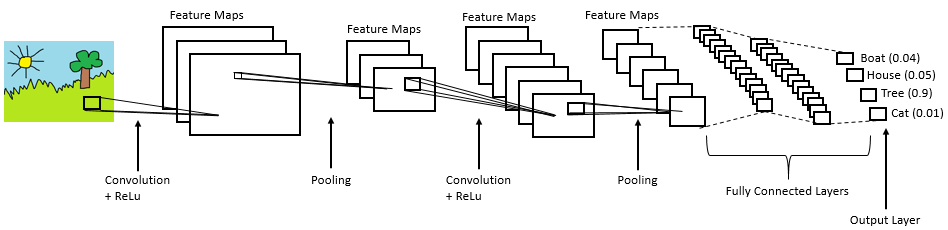
---TRIPLE TROUBLE

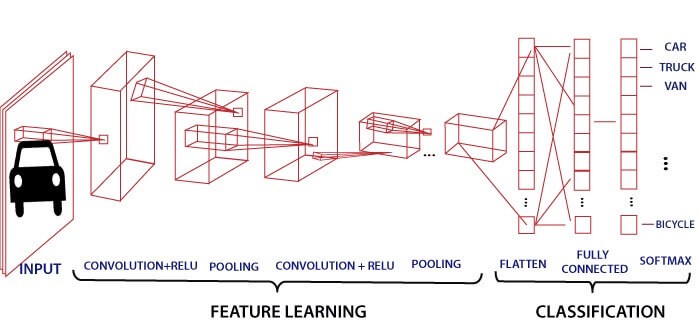


**Self-Driving Car** is an autonomous vehicle that senses the surroundings and drives the car without Human input. Here in this project we followed the exact same motive and trained a car to autonomously drive on a race track using **Carla Simulator**. **CARLA** is an open-source autonomous driving **simulator**. It was built from scratch to serve as a modular and flexible API to address a range of tasks involved in the problem of autonomous driving. Due to Covid we weren’t able to perform the task physically hence we used a virtual environment. There are various maps that are provided by Carla, we here specifically used the race track.



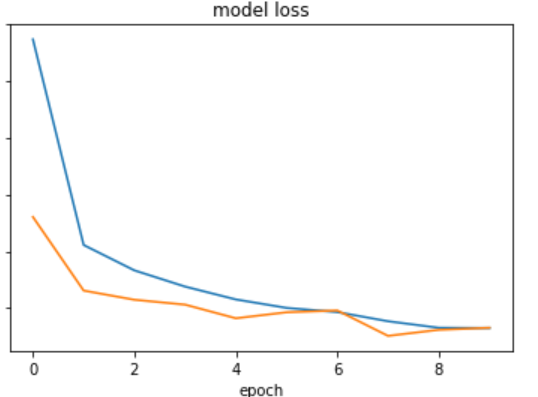
We are curious that how a particular car is hence trained in order to accomplish the goal that it runs by itself. Well, there are networks through which the data is passed. There are various Neural Networks we can use based on the kind of data we use. We used CNN model here. **Convolutional neural networks** (CNN) are all the rage in the deep learning community right now. These CNN models are being used across different applications and domains, and they’re especially prevalent in image and video processing projects. CNN learns the filters automatically without mentioning it explicitly. These filters help in extracting the right and relevant features from the input data. CNN captures the spatial features from an image. Spatial features refer to the arrangement of pixels and the relationship between them in an image. They help us in identifying the object accurately, the location of an object, as well as its relation with other objects in an image.





The whole process involved----

* We started by setting up Carla in our laptops and then creating a virtual environment and worked on a basic Server Client mode. We even installed all the packages and modules required for the project that is for the code we were required to write for collecting the data to training our data and creating a model and then using that model. Later on, we edited the code given for Data Collection (datacollection.py) in such a way that with one single press on the button W, the throttle would be set to 0.35 and then with continuous use of that button it would increase with a factor of 0.25 but would never cross 2 (highest attained value will be 2.1), similarly with the button A and D the steering would be set to -0.25 or +0.25 initially and then both the values would increase if the button is pressed again but the values would never cross -1 and 1. We also do have a break function activated by S which will turn the throttle back to 0. There was a button F which was basically for recording the data that is it would take snaps from the video of our driving from that data.
* We collected the data in the ‘img’ folder of PythonClient ( a folder inside the CarlaSimulator folder that contained out codes ) and then even used the code of viewing the collected data to load it.
* Further we started our model training on the Google collab Jupyter (a platform that would allow the code to be written in a proper way and provide us a space to run our code there ) and there we converted our mat file (“data”- that had all our stored data that was to be treated) into a csv file (to train out data) so that it can be utilised in variable “lap”, which is a directory containing images as “values” and its steer , throttle as “key” , later we extracted the steer and throttle value and then augmented our data (by choosing the type and number of images we wanted). Now we used a CNN model provided by nivedia for our architecture (where we also used Dropout(a function used that decides how much data you want to use in CNN in that particular layer coming from the previous layer ) and BatchNormalisation (to avoid high variance) ) , it had total of 6 layers , later on we saved our trained model after training our data through 20 epochs (20 times it was trained and passed through the whole CNN ), we even use several functions like Early Stopping (if the loss after about x (defined in there) epochs doesn’t decrease then it stops the epochs there itself) to make our training and validation loss less and to get our model more correctly trained.
* A very important aspect is that for predicting the throttle and steer we used the same model but different model training and hence it went to different model trained files (‘model\_th1.h5’ for steer and ‘model\_tt1.h5’ for throttle) and we used the value predicted by these in our datacollection file.
* Later on we created a file called “model.py” that could store our architecture and help to drive our car autonomously on Carla. And there we passed the path to our model in load\_weights , then we edited the “datacollection.py” to provide the facility to self-drive the car by pressing P wherein both throttle and steer would be determined by our trained model.



**The loss curve during our model training**

The loss curve is something that compares the output that it gains from the whole network and trained model and the output it should have had gained(that is the measurement of the vectors like speed, steer , etc that was of that particular image when the data was collected). This as expected should be as less as possible and we have tried our best to do so.