SELF DRIVEN CAR

1 Mrudula Oruganti, 2 Abhishek G, 3 Kaushik Kannan, 4 Shwetik Thakur, 5 Prahitya Mahavir

1*Mrudula Oruganti Guide, Computer Science & Engineering, SRM University Chennai,India*
2 Abhishek G Student, Computer Science & Engineering, SRM University Chennai, India
3 Kaushik Kannan Student, Computer Science & Engineering, SRM University Chennai, India 4 Shwetik Thakur Student, Computer Science & Engineering, SRM University Chennai, India
5 Prahitya Mahavir Student, Computer Science & Engineering, SRM University Chennai, India

ABSTRACT

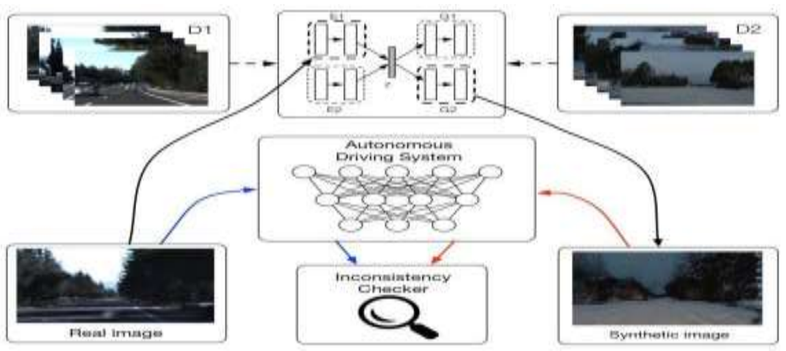
An adversarial network is a deep learning framework that makes use of multiple deep learning network as "adversaries" critiquing the results generated to maximize the probability of generative results. This can be employed in training a Self Driving Car. This cannot be embedded in a self driving car until fully trained in a simulated environment. The self driving car and it's sensors are to be operated in a virtual environment. The model is so trained that an actor critic model M trains to manoeuvre the vehicle with a reward based system. A discriminator network uses it's pre-trained models to critique the efficiency of the self driving car. The car is set in obstacle courses which it is to avoid. The unsupervised learning process if critiqued using a discriminator like a mentor to the network, until the self driving car learns to avoid all obstacles. Further the camera sensors of the car using YOLO(You only look once algorithm) to detect objects and noise based transformations are applied to understand the the range of objects to avoid and signs to read. Convolution neural networks are used for the YOLO algorithm to actively distinguish objects to avoid, road unevenness to reduce speeds, speed breakers to tackle and unpredictable human behaviour to account for

Keyword.:- *adversarial, probability, Self Driving Car, sensors, manoeuvre, obstacle, YOLO, Convolution neural networks*

1 EXISTING MODULE

While Deep Neural Networks (DNNs) have established the fundamentals of DNN-based autonomous driving systems, they may exhibit erroneous behaviors and cause fatal accidents. To resolve the safety issues of autonomous driving systems, a recent set of testing techniques have been designed to automatically generate test cases, e.g., new input images transformed from the original ones. Unfortunately, many such generated input images often render inferior authenticity, lacking accurate semantic information of the driving scenes and hence compromising the resulting efficacy and reliability. In this paper, we propose Deep CNN, an unsupervised framework to automatically generate large amounts of accurate driving scenes to test the consistency of DNN-based autonomous driving systems across different scenes.

In particular, Deep CNN delivers driving scenes with various weather conditions (including those with rather extreme conditions) by applying the Generative
Adversarial Networks (GANs) along with the corresponding real-world weather scenes. Moreover, we have implemented Deep CNN to test three well-recognized DNN-based autonomous driving systems. Experimental results demonstrate that Deep CNN can detect thousands of behavioral inconsistencies for these systems.



Based on our assumption, an autonomous driving system is consistent if its steering angle prediction does not change after modifying the weather condition of driving images. However, this assumption is too strong to be practical since minor steering angle change incurred by the scene change may still fall into the safe zone. Hence, similar with prior work [28], we relax the assumption and accept the prediction if the difference between the predicted steering angles of original and transformed images can be within an error bound.

2. PROPOSED MODEL

We propose a conceptually simple and lightweight framework for deep reinforcement learning that uses asynchronous gradient descent for optimization of deep neural network controllers. We present asynchronous variants of four standard reinforcement learning algorithms and show that parallel actor-learners have a stabilizing effect on training allowing all four methods to successfully train neural network controllers. The best performing method, an asynchronous variant of actor-critic, surpasses the current state-of-the-art on the Atari domain while training for half the time on a single multi-core CPU instead of a GPU. Furthermore, we show that asynchronous actor-critic succeeds on a wide variety of continuous motor control problems as well as on a new task of navigating random 3D mazes using a visual input.

2.1 Systematic Testing with Neuron Coverage

The input-output space (i.e., all possible combinations of inputs and outputs) of a complex system like an autonomous vehicle is too large for exhaustive exploration. Therefore, we must devise a systematic way of partitioning the space into different equivalence classes and try to cover all equivalence classes by picking one sample from each of them. In this project, we leverage neuron coverage as a mechanism for partitioning the input space based on the assumption that all inputs that have similar neuron coverage are part of the same equivalence class (i.e., the target DNN behaves similarly for these inputs). Neuron coverage was originally proposed by Peietal. for guided differential testing of multiple similar DNNs . It is defined as the ratio of unique neurons that get activated for given input(s) and the total number of neurons in a DNN:

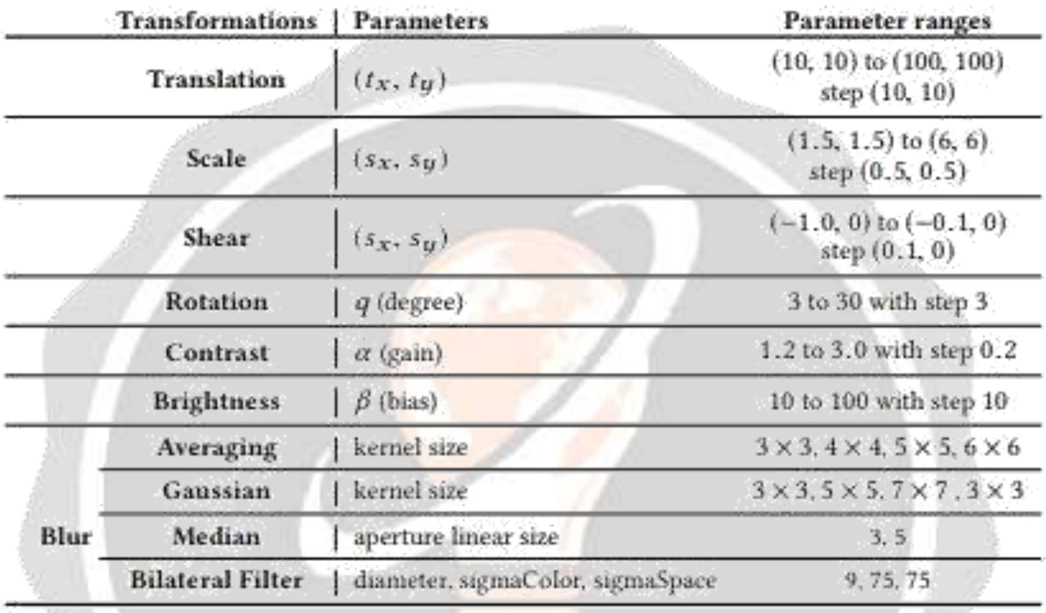
$$\text{Neuron Cover are} = |\text{Activated Neurons}| / |\text{Total Neurons}|$$

An individual neuron is considered activated if the neuron’s output (scaled by the overall layer’s outputs) is larger than a DNN-wide threshold. In this paper, we use 0.2 as the neuron activation threshold for all our experiments. Similar to the code-coverage-guided testing tools for traditional software, DeepTest tries to generate inputs that maximize neuron coverage of the test DNN.

For all neurons in fully-connected layers, we can directly compare their outputs against the neuron activation threshold as these neurons output a single scalar value. By contrast, neurons in convolutional layers output multidimensional feature maps as each neuron outputs the result of applying a convolutional kernel across the input space . In such cases, we compute the average of the output feature map to convert the multidimensional output of a neuron into a scalar and compare it to the neuron activation threshold.

2.2 Transfer Learning

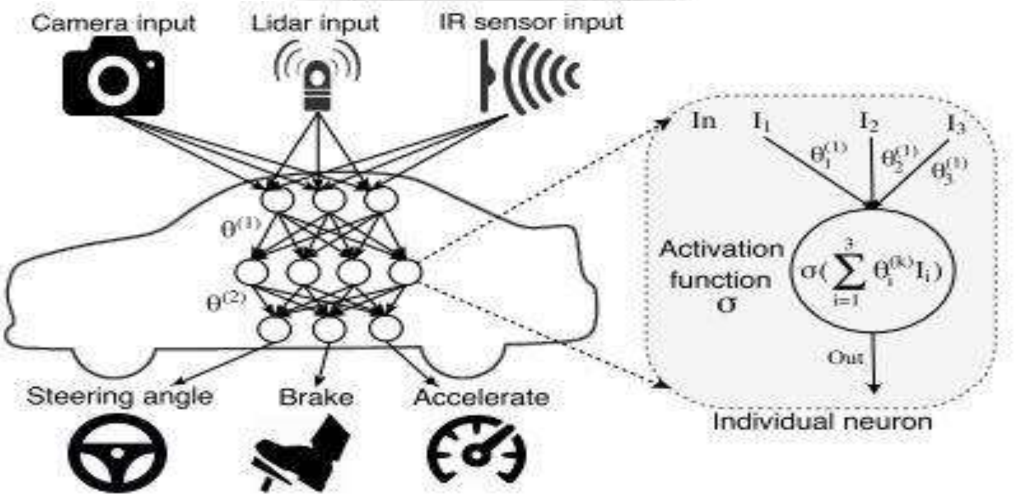
For this model, we used the idea of transfer learning. Transfer learning is a way of using high quality models that were trained on existing large datasets. The idea of transfer learning is that features learned in the lower layers of the model are likely transferable to another dataset. These lower level features would be useful in the new dataset such as edges. Of the pre-trained models available, ResNet50 had good performance for this dataset. This model was trained on ImageNet. The weights of the first 15 ResNet blocks were blocked from updating (first 45 individual layers out of 175 total). The output of ResNet50 was connected to a stack of fully connected layers containing 512, 256, 64, and 1 different units respectively. The architecture of this model can be seen with the overall number of param.eters being 24,784,641. The fully connected layers used ReLUs as their activation. The ResNet50 model consists of several different repeating blocks that form residual connections. The number filters varies from 64 to 512. A block is consistent of a convolutional layer, batch normalization, ReLU activation repeated three times and the input layer output combined with the last layer. Other sizes of locking were attempted, but produced either poor results or were slow in training. For example, 3 training only the last 5 blocks provided poor results, which were only slightly better than predicting a steering angle of zero for all inputs. Training all the layers also produced worse results on the validation set compared to blocking the first 45 (0.0870 on the validation set after 32 epochs). The model took as input images of 224x224x3 (downsized and mildly stretched from the original Udacity data). The only augmentation provided for this model was mirrored images. Due to the size constraints of the input into ResNet50, cropping was not used as it involved stretching the image. The filters in the pretrained model were not trained on stretched images, so the filters may not activate as well on the stretched data (RMSE of 0.0891 on the validation set after 32 epochs). Additionally, using the left and the right cameras from the training set proved not to be useful for the 32 epochs used to train (0.17 RMSE on the validation set).



Transformations		Parameters	Parameter ranges
Translation		(t_x, t_y)	(10, 10) to (100, 100) step (10, 10)
Scale		(s_x, s_y)	(1.5, 1.5) to (6, 6) step (0.5, 0.5)
Shear		(s_x, s_y)	(−1.0, 0) to (−0.1, 0) step (0.1, 0)
Rotation		q (degree)	3 to 30 with step 3
Contrast		α (gain)	1.2 to 3.0 with step 0.2
Brightness		β (bias)	10 to 100 with step 10
Blur	Averaging	kernel size	3×3, 4×4, 5×5, 6×6
	Gaussian	kernel size	3×3, 5×5, 7×7, 3×3
	Median	aperture linear size	3, 5
	Bilateral Filter	diameter, sigmaColor, sigmaSpace	9, 75, 75

2.3 Deep Neural Networks incorporarion with the Hardware

A typical feed-forward DNN is composed of multiple processing layers stacked together to extract different representations of the input [30]. Each layer of the DNN increasingly abstracts the input, e.g., from raw pixels to semantic concepts. For example, the first few layers of an autonomous car DNN extract low-level features such as edges and directions, while the deeper layers identify objects like stop signs and other cars, and the final layer outputs the steering decision (e.g., turning left or right).



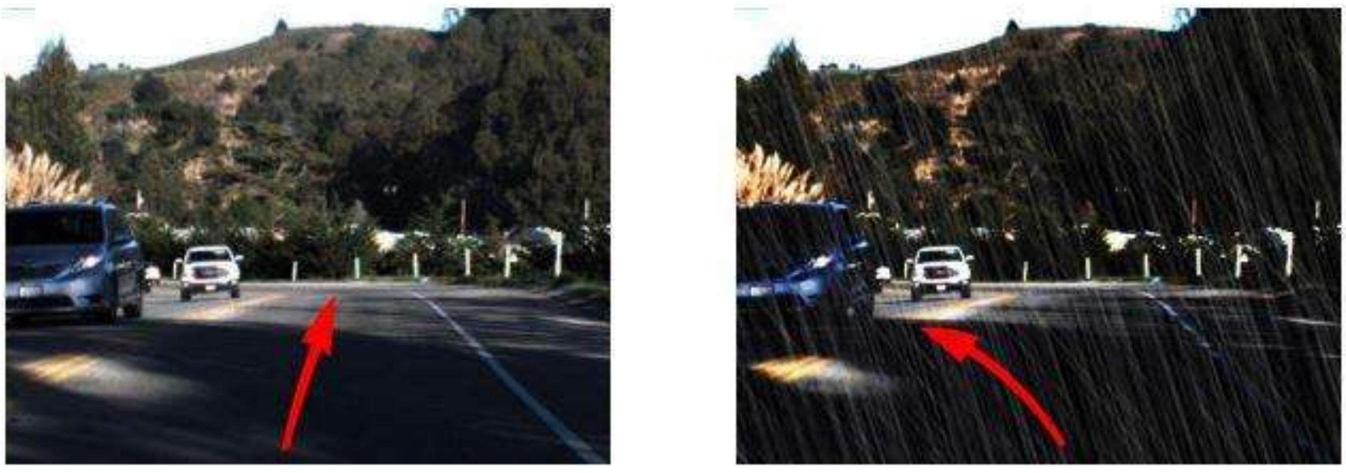
Each layer of a DNN consists of a sequence of individual computing units called neurons. The neurons in different layers are connected with each other through edges. Each edge has a corresponding weight.Popular activation functions include ReLU (Rectified Linear Unit) , sigmoid, etc. The edge weights of a DNN is inferred during the training process of the DNN based on labeled training data. Most existing DNNs are trained with gradient descent using backpropagation. Once trained, a DNN can be used for prediction without any further changes to the weights. For example, an autonomous car DNN can predict the steering angle based on input images.

2.4 Data Collection

We use a real-world dataset released by Udacity as a baseline to check the inconsistency of autonomous driving systems. From the dataset, we select two episodes of highway driving video where obvious changes of lighting and road conditions can be observed among frames. To train our UNIT model, we also collect images of extreme scenarios from Youtube. In the experiments, we select snow and hard rain, two extreme weather conditions to transform real-world driving images. To make the variance of collected images relatively large, we only search for videos which is longer than 20mins.

3 Future Work

Tests generates realistic synthetic images by applying different image transformations on the seed images. However, these transformations are not designed to be exhaustive and therefore may not cover all realistic cases. While our transformations like rain and fog effects are designed to be realistic, the generated pictures may not be exactly reproducible in reality due to a large number of unpredictable factors, e.g., the position of the sun, the angle and size of the rain drops. etc. However, as the image processing techniques become more sophisticated, the generated pictures will get closer to reality. A complete DNN model for driving an autonomous vehicle must also handle braking and acceleration besides the steering angle. We restricted ourselves to only test the accuracy of the steering angle as our tested models do not support braking and acceleration yet. However, our techniques should be readily applicable to testing those outputs too assuming that the models support them.



1.1 original 1.2 with added rain

These models are far from perfect and there is substantial research that still needs to be done on the subject before models like these can be deployed widely to transport the public. These models may benefit from a wider range of training data. For a production system, a model would have to be able to handle the environment in snowy conditions. Generate adversarial models, GANs, could be used to transform a summer training set into a winter one. Additionally, GANs could be used to generate more scenes with sharp angles. Additionally, a high quality simulator could be used with deep reinforcement learning. A potential reward function could be getting from one point to another while minimizing time, maximizing smoothness of the ride, staying in the correct lane/following the rules of the road, and not hitting objects.

4.CONCLUSION

In this paper, we propose Deep CNN, an unsupervised GANbased approach to synthesize authentic driving scenes with various weather conditions to test DNN-based autonomous driving systems. In principle, Deep CNN applies the metamorphic testing methodology to detect the inconsistent autonomous driving behaviors across different driving scenes. The experimental results on three real-world Udacity autonomous driving models indicate that Deep CNN can successfully detect thousands of inconsistent behaviors. Furthermore, our results also show that Deep CNN can be promis-ing in measuring the robustness of autonomous driving systems. Currently, Deep CNN only supports two weather conditions, we plan to support more weather conditions to fully test autonomous driving systems under various conditions in the near future.