Abstract

As long the automobiles have existed, self-driving cars have been a dream. As we consider domains to which we can apply intelligent systems, the automotive industry stands out as having the most potential for impact. Safety is one of the major reason autonomous vehicles must be developed. Average time people spent in driving in driving is high and that same time can be spent doing other constructive activities. Even for people who have lost driving privileges owing to age and other disabilities, autonomous vehicles can prove to be a boon for such people and can make life a bit easier. This would also increase the number of people who would consider buying automobiles and would cause an increase drift in the number of vehicles.

Another benefit that autonomous vehicles will cause is the huge impact it will cause in saving the fuel intelligently. Human drivers tend to commit error in managing the speed according to the environment and often this leads to the mismanagement of the fuel resources. Intelligent droving option would result in optimized use of the resources for the vehicle. This will also be a great step towards environment where we can save loads of natural resources in the driving industry.

1 Introduction

Self-driving cars can improve and exceed the human ability to drive and help in saving thousands of lives each year. A steep increase in computational capabilities in recent years has boosted the growth in various autonomous driving attempts. The increased ability to train deep neural network have played a crucial role in self driving cars helping it to make important real-time decisions accurately. the machine learning algorithms are extensively used to find the solutions to various challenges arising in self-driving cars. With the incorporation of various sensors coupled with the machine learning techniques self-driving system can accomplish new tasks. The existing approaches in semi-autonomous driving cars help the drivers with many options such as blind spot monitoring, pedestrian detection, adaptive cruise control and lane departure warning. We plan to build a system that will lead to better performance of the self-driving cars. The motivation of the project is to create a system that could self-learn to drive in each environment. We narrowed this and focused our aim to predict steering angle using Convolutional Neural nets. We propose to build a system that learns to drive by

observing. Using the power of neural networks, we want to approach the problem of self-driving cars by predicting the steering angle as the training signal and output.

2 Challenges

One of the major challenges that autonomous driving cars face are that they need to be made robust enough to take fast and quick decisions in diverse and unpredictable situations which may include moral dilemmas. As expected they must be made environmental friendly and take optimized decisions case of all resources. These factors make deep neural networks indispensable for such difficult tasks. Another major challenge in development of self-driving cars is to meet the government regulations and match the safety standards that are set up. Due to some of the recent mishaps in self-driving cars, some states have banned the use of self-driving cars and questioned the viability of the existence of the cars. Taking the safety factors in consideration the system of self-driving cars must be built robust and must include the ability to handle all the critical tasks that can arise in a driving situation. Such system needs to constantly face conflicting situations while programming example sleeping passenger is placed at a priority that may lead to reduce the speed and jam the traffic behind and crash risks. This leads to many aspects like fairness and regulations which questions the viability of self-driving cars. Another Challenge is to find the perfect model which would accommodate the accuracy of human brain and perform perfectly with zero error in all situations. The computation power in making such model work and the complexity still needs to be improved so that the self-driving car does not remain a wonder. Other factors like cost and testing such vehicles when they will be mass produced remains as a huge challenge for the makers of the self-driving cars.

3 Current Trends and Related Work

There have been tremendous recent advances in self-driving cars. The major players in this is Waymo by Google. This self-driving attempt is based on LIDAR system which sends out infrared rays multiple times per second and observes the time these rays take to reflect and come back. Based on

this information it creates a real time 3D map which detects and classifies the objects in surrounding. Based on this the computer in the car takes real time decisions about driving. Another such attempt is Autopilot feature by Tesla which assist in driving with the cruise control option using a RADAR system and 12 sensors mounted on the vehicle. Another major and recent contribution is by the MIT CSAIL team which helps in navigation of autonomous vehicles in area with unreliable maps, unlit roads and unclear road lines. The team has developed an interesting approach which feeds the system with real time map of the surrounding known was Map Lite. ALVINN is the first attempt in self-driving cars using a neural net. It was developed by Dean Pomerleau in 1989 sing a single fully connected layer. Another such attempt was by NVIDIA exploited the CNN abilities to classify complex scenes which are present in driving environment. Other approaches which have gained momentum is usage of RNN which includes the sequential input rather than random and mimics the environment closely. Deep Reinforcement learning is a reward based system which can also help in taking critical decisions resulting into minimum loss in case of casualties. Another early attempt in self driving cars is the 2004 Volkswagen Touareg R5, and a 2006 Volkswagen Passat Wagon, respectively. Both vehicles utilize custom interfaces to enable direct, electronic actuation of throttle, brakes, gear shifting, and steering. Vehicle data are communicated to Linux-based computer systems through CAN bus interfaces. Another such attempt in self-driving cars was Boss, a modified 2007 Chevy Tahoe won the Darpa Urban challenge using a combination of laser, radar, and GPS data to safely navigate a 53-mile test course among 60 other vehicles (10 autonomous and 50 human-driven). The challenge called for autonomous vehicles to drive 60 miles through an urban environment, interacting with other moving vehicles and obeying the California Driver Handbook

4 Overview of the simulated system used for data collection

The training data was collected using Udacity's simulator, in training mode. The Simulator works same as DAVE 2 Model of Nvidia. It can produce 3 images just like from the 3 cameras mounted in the Car that is Left, Center and Right. We trained on the first track, using keyboard input. For

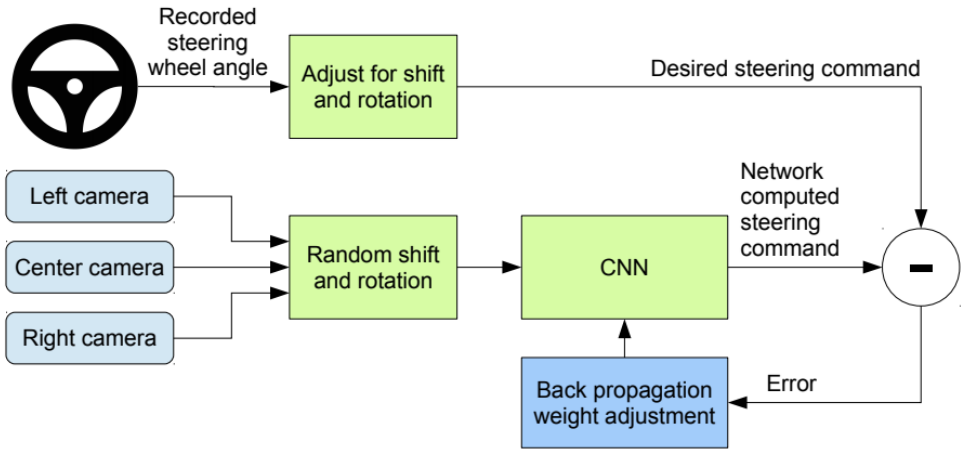

Figure 1: High-level view of the data collection system

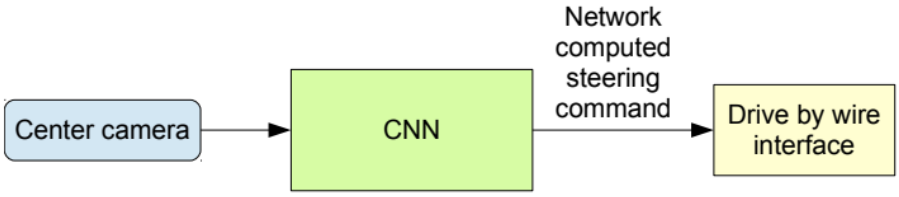

Figure 2: The trained network is used to generate steering commands from a single front-facing center camera.

normal driving data, we drove 2 laps around the track, in the default direction (counter-clockwise looking from the top) . A block diagram of the training system we use is shown in Figure 1. Images are fed into a CNN which then computes a proposed steering command. The proposed command is compared to the desired command for that image and the weights of the CNN are adjusted to bring the CNN output closer to the desired output. The weight adjustment is accomplished using back propagation.

Once trained, the network can generate steering from the video images of a single center camera. This configuration is shown in Figure -2.

5 Design and Implementation

Since the task of predicting steering angles from an input image is a computer vision task, a convolutional neural network (CNN) architecture is most appropriate. The most significant difference between a CNN and a fully connected DNN is the presence of a convolution layer. The neurons in a convolution layer are connected only to some of the neurons in the next layer and multiple connections among different neurons share the same weight. The sets of connections sharing the same weights are essentially a convolution kernel that applies the same convolution operation on the outputs of a set of neurons in the previous layer. Convolution layers have two major benefits. First, they greatly reduce the number of trainable weights by allowing sharing of weights among multiple connections and thus significantly cut down the training time. Second, the application of convolution kernels is a natural fit for image recognition as it resembles the human visual system which extracts a layer-wise representation of visual input.

Initially we trained two different promising network architectures to see if we was able to measure a significant difference in performance between the two. The first is the architecture from NVIDIA's 2016 paper "End to End Learning for Self-Driving Cars", the second is a modified version of theComma.ai model. For this particular application We were not able to measure a significant difference in performance between the two architectures and got good results with both, so we stuck with the latter. While the network architecture is always important, as long as you pick one of many possible suitable architectures, the data you collect and the data augmentation techniques you apply are the more crucial and more difficult parts to solve this problem. Hence, the majority of the work here focuses on the collection and processing of the training data rather than on model hyper parameter tuning.

We trained the weights of our network to minimize the mean squared error between the steering command output by the network and the command of either the human driver, or the adjusted steering command for off-center and rotated images. The general CNN network architecture used for training the model is shown in Figure 4- The comma.ai CNN model we used to train our model consists of the following layers The architecture of the final model is as follows

- RGB image input with dimensions 80x160x3

- Keras Cropping2D layer to crop the input to 50x150 pixels
- Keras Lambda layer to convert the feature value range to [-1,1]
- Three convolutional layers with 32, 64, and 128 filters, filter sizes 8x8, 5x5, and 3x3, and strides 4, 2, and 2.
- One dense layer with 512 units following the convolutional layers, and one output unit
- ELUs as nonlinearities after each layer except the output unit
- Batch normalization after each layer except the output unit
- Dropout after the third conv and first dense layer (both rate 0.5)

Instead of following the conv layers with max pooling layers we used a convolutional stride greater than 1, namely 4 for the first conv layer and 2 for the second and third conv layers. we didn't do this because we had good theoretical reasons for it, but because the Comma.ai model that our model is based on did it this way and we considered it an experiment to see if it would produce good results. Intuitively, reducing complexity through pooling should be superior over reducing complexity by increasing the stride in the conv layers, since the former method chooses the most relevant information out of all possible filter positions, while the latter method loses information by skipping part of the possible filter positions to begin with. In practice, however, it seems to work well enough. We used ELUs are they are strictly strictly superior over than ReLUs. Batch normalization helps reduce overfitting and a mean squared error loss function is used. The model uses an Adam optimizer with the default learning rate of 0.001 and a decay of 5e-05. Training will stop early and the learning rate will be reduced in case of the validation loss plateauing for several epochs in a row. Here is a visualization of the architecture. The default visualization that comes with Keras is not exactly pretty, but it shows the layer dimensions:

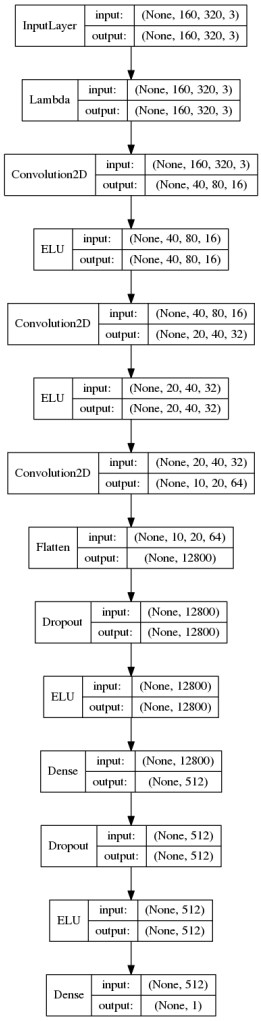

Figure 3: CNN Architecture

6 Optimization Techniques

We use Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning

Adaptive Moment Estimation (Adam) is the optimization method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients, Adam also keeps an exponentially decaying average of past gradients. The decaying averages of past and past squared gradients are computed using the estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients. As these estimates are initialized as vectors of 0's, they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small. These biases are counter-acted by computing bias-corrected first and second moment estimates and then used to update the network parameters. To make the architecture more robust and to prevent over fitting dropout layers are added to the network as shown in figure 4. Dropout disables neurons in your network by a given probability and prevents co-adaption of features.

7 Experiment Evaluation

7.1 Data Collection and Preprocessing

The simulator records image data from three cameras, one center camera and one camera on each the far left and right sides of the car, recording 10 images per second as shown in figure5. The images from the two non-center cameras simulate the effect of the car being too far left or too far right in the lane and by adding or subtracting an appropriate offset to/from the respective center camera steering angle, one can effectively produce artificial recovery data.

Note that we deliberately did not record any recovery data, i.e. we did not record any data of the car correcting its course from the edges of the

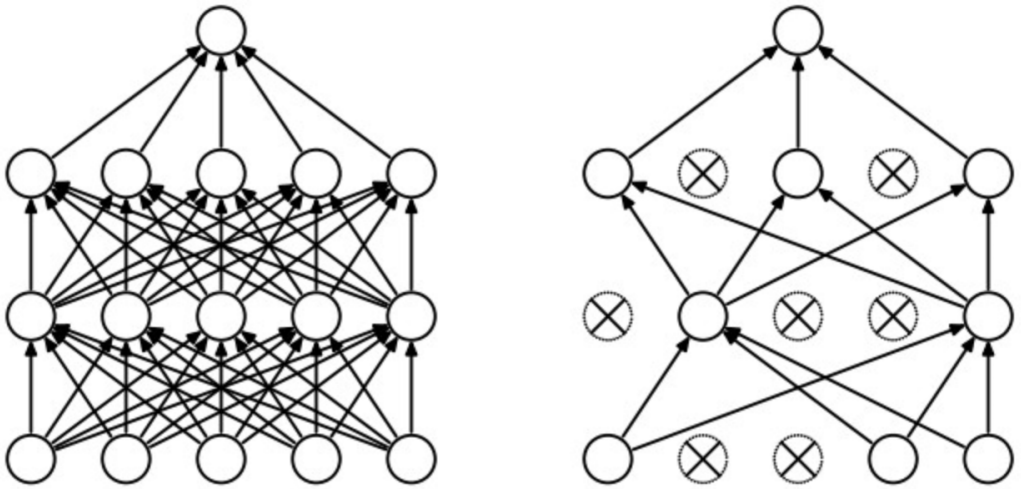

Figure 4: Dropout in CNN

lane back towards the center. Since real cars on real roads cannot really make use of this technique and can still learn how to drive autonomously, our model should be able to learn without such data, too. Instead we used the data from all three cameras for the training and hoped that the left and right camera images and some geometric transformations of the images would be enough to produce the same effect that recovery data would, which turned out to be true. Not to mention that it is a lot more efficient than recording lots of manually produced recovery data. We recorded roughly two laps (maybe it was a bit less) of good driving behavior in the default direction (counter-clockwise) on the lake track (track 1), followed by roughly four laps in the reverse direction (clock-wise). We ended up with a little more than 24,000 images when we was done recording, i.e. around 8,000 per camera. We reduced the original size of the recorded images (160x320 pixels) by half in both dimensions to 80x160 pixels and then cropped 20 pixels at the top and 10 pixels at the bottom because they only contain the sky and the hood of the car - visual information that is irrelevant to predict the steering angle. We also cropped 5 pixels each on the left and right for the same reason. It might be useful to crop even more pixels from the top to eliminate even more irrelevant or even misleading image information, but We got satisfactory results with this processing.

Figure 5: Images from 3 Camera angles

7.2 Steering Angle Adjustment for the Left and Right Camera Images

The images from the two non-center cameras unsurprisingly turned out to be very useful to train off-center recovery. Experiments with different steering angle adjustments for the left and right cameras by adding/subtracting constants ranging from 0.1 to 0.25 yielded adjustments of 0.15-0.2 to be reasonable values. We also experimented with non-constant adjustment values that depend on the magnitude of the center camera steering angle, the reasoning being that the larger the curve radius, the more time the car has to revert back towards the center of the lane, allowing for smoother corrections, while the sharper the curve, the faster the car has to revert back toward the center. By contrast, if the angle adjustment is an additive constant, the correction back to the center of the lane is always equally fast (which means equally abrupt), regardless of the curvature of the road. We ended up discarding the magnitude-dependent approach though, since it introduced more complexity for unclear gain. Of course non-center camera images are just two specific cases of horizontal translation, and as will be described in the next section, We applied horizontal translation randomly to the entire training dataset to generate many more different viewpoints.

7.3 Data Augmentation

Data augmentation is essential to solve this problem, training on data of good driving behavior alone will not result in a working model. At the same time data augmentation is also more complex in this case than in a classification task, since for many relevant transformations of the input data, the corresponding labels need to be adjusted in a non-trivial way. A bear is a bear whether you flip the image or not, but the steering angle of the perspectively distorted image of a road might need to be adjusted in a non-

obvious way. Figuring out how exactly to adjust the steering angle for some transformations turns into a project of its own, and a lot of work goes into it. Below we describe the transformations we experimented with and my findings regarding which transformations worked or didn't work, which were useful or unnecessary, and what steering angle adjustments turned out to work well. We tested the following image transformations:

- Flipping images horizontally to prevent a bias towards being able to handle some situations only in one direction but not the other. The steering angle is being inverted (additive inverse) accordingly.

- Changing the brightness, particularly decreasing it, to make the model less dependent on certain colors, to make it recognize lane markings with less contrast, and to cater to the darker colors of the mountain track.

- Three kinds of transformations came to our mind as possible candidates to recover from off-center positions and to ensure that the model can handle sharp curves: Rotation, horizontal translation, and a perspective transform simulating a change in the curvature of the road. We tested the effectiveness of all three .

- Transforming the perspective to simulate an incline change uphill or downhill. The purpose of this was to use the data from the flat lake track to train the model for the mountain and jungle tracks, both of which contain many slope changes. Results of our data augmentation experiments:

- Horizontal flipping: This one is a no-brainer - unsurprisingly it helps a lot and should always be applied (randomly to half of your data).

- Changing the brightness: It had exactly the desired effect. Thanks to decreasing the brightness of the lake track images, the model was able to drive on the much darker mountain track without ever having seen it during training. Depending on the training iteration, We randomly varied the brightness of 10-50percent of the images between factor 0.4 and 1.5 of the original brightness.

- Translation: Horizontal translation is just an extension of the effect of using the left and right camera images and is very helpful, if not

essential, to training a model that stays close to the center of the lane. We randomly translated the images by 0 to 40 pixels, sometimes 0 to 50 pixels. Steering angle adjustments of 0.003-0.004 per pixel of translation turned out to yield reasonable correction speeds that are neither too abrupt on straight roads nor too slow in sharp curves. Vertical translation turned out to be unnecessary. We did it a little bit (0-10 pixels) just to create more diverse data, but vertical translation does not serve as an even remotely realistic proxy for simulating changes in the slope of the road.

- Curvature perspective transform: This turned out to be useful to simulate sharper curves on the one hand, but even more importantly it simulates situations in which the car is oriented at an angle to the lane rather than parallel to the lane. The image above illustrates this effect. If you compare the central vertical grid line in the original image and the distorted image you see that the distorted image simulates the car being oriented toward the side of the lane rather than toward the center of the road as in the original image. Of course, this primitive perspective distortion is a very imperfect proxy for a change in the curvature of the road. To truly increase the sharpness of a curve in a realistic way for example, one can of course not just shift the pixels in the linear way that this transform does, but this approximation still did an alright job.

- Rotation: We experimented with rotating images to simulate a change in the curvature of the road, but in most cases, this does not yield a realistic approximation, and more importantly it is inferior to the perspective transform described above. We did not end up using this transform.

- Incline perspective transform: While it generally actually is a more realistic approximation than the curvature transforms above, it turned out to be completely unnecessary - We did not end up using this.

The function that actually applies these transformations is the generator function defined In a nutshell, it loads batches of training data and labels, applies the transforms specified in the arguments, yields the results, shuffles the dataset upon each complete pass, and can do this indefinitely. Each transform has its own independent application probability, and some can

choose from a number of modes to operate in. The generator function provides some options to apply the above image transforms in a more targeted way. For example, for the curvature transformation, the mode argument specifies whether all images are eligible for a given transform, or only images with a certain minimum or maximum corresponding absolute steering angle, or only to images with a corresponding steering angle that is positive or negative. During training, it sometimes proved helpful to apply the curvature transform only to images of an already curved road. It was also better to apply the artificial curvature only in the same direction as the original curvature. The likely reason for this phenomenon is that the steering angle adjustment associated with the artificial curvature change is not chosen perfectly, and if a road that was curved to the left is artificially straightened by being transformed to the right does not end up with the appropriate steering angle (e.g. zero), then this creates conflicting training data. Note that assemble filelists() returns the steering angle list as a list with two columns, containing not only the steering angle, but also the original steering angle of the center camera version of the respective image. The reason for this is that the original center camera steering angle is a reasonable indicator for the actual curvature of the road (assuming that We drove relatively cleanly along the trajectory of the road) while the adjusted steering angles of the left and right camera images are not. Example: If an image has a steering angle of -0.15, it might be a slight left turn, but it might also be the right camera image of a straight part of the road (or neither). Hence it is useful to preserve the original steering angle associated with the center camera image for all images. The mode option in the generator function uses this original center camera steering angle to decide which images are eligible for transformation an which aren't.

8 A word on Validation

We did use a validation dataset, but while the validation error is helpful to monitor overfitting and to make sure that your model is getting better, it is not the crucial metric to look at here. Your model either can or cannot drive the car around the entire track, and the validation error can't tell you when that point is reached (if your validation dataset even reflects all relevant situations!). Consider this: Whether your model predicts slightly incorrect steering angles in a lot of situations or a severely incorrect steering angle in

Figure 6: Experimental Results

only one situation might result in roughly the same validation error, but in the former case your car might make it around the track and in the latter case it will drive off a cliff. And if, in the latter case, that one situation where your model fails badly is not reflected in your validation dataset, then you might even get a near-zero validation error despite the model failing. The bottom line is, the validation error played only a small role for the decisions We made, the crucial and more insightful test is to watch your model drive in autonomous mode.

9 The Training and Test Process

With some data augmentation it was possible to get the model to drive well. 10-12 epochs on 90percent (the remaining 10percent were held out for validation) of the 24,000-image dataset recorded on the track were enough to achieve this. The trained model was stored as .h5 file. We tested our model on the autonomous mode of the Udacity Simulator. The images of the vehicle driving by itself and its respective angles and throttle are shown in figure6.

10 Cons

Of course, the model could still get a lot better: It often over-compensates when it recovers from the edge of the lane. Part of this suboptimal behavior starts with the training data we recorded: Trying to input very precise steering commands with a keyboard, mouse or game controller makes you appreciate what an amazingly precise input device a real steering wheel is. It is very difficult to stay in the center of the lane at all times, and We don't need to mention that We could absolutely not manage to do that. Another reason for flaws in the model's behavior are flaws in our image transformations: The only geometric transformation that does not degrade the quality of the input data is the horizontal flip. All other above geometric transformations are flawed approximations of the real phenomena they are trying to simulate, and our steering angle adjustments, however carefully chosen, are flawed approximations of the ideal steering angle adjustments

11 Conclusion and Future Work

We have shown that it is possible to create a model that reliably predicts steering wheel angles for a vehicle using a deep neural network and a plethora of data augmentation techniques. While we have obtained encouraging results, we would like in the future to explore the following:

- Take into account speed and throttle in the model
- Get the car to drive faster than 20MPH
- Experiment with models based VGG/ResNets/Inception via transfer learning
- Use Recurrent Neural Networks to Udacity dataset
- Experiment with Reinforcement Learning

As can be seen, there are many areas we could explore to push this project further and obtain even more convincing results. One of the most important learnings from this project is the importance of DATA ,without all those images and steering angles, along with their potentially infinite augmentations, we would not have been able to build a robust enough model.

12 References

- M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316, 2016.

- M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. arXiv preprint arXiv:1704.07911, 2017.

- A. El Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. Autonomous Vehicles and Machines, Electronic Imaging, 2017.

- A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in neural information processing systems, 2012, pp. 1097–1105.

- Sebastien C. Wong, Adam Gatt Victor Stamatesc "Understanding data augmentation for classification"

- G.Huang,Z.Liu,K.Q.Weinberger,andL.vanderMaaten. Densely connected convolutional networks. arXiv preprint arXiv:1608.06993, 2016.

- S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.

- M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in ECCV, 2014.

- Net-Scale Technologies, Inc. Autonomous off-road vehicle control using end-to-end learning, July 2004.

- Udacity. An open source self-driving car, 2017.

- https://github.com/commaai/research

- https://github.com/udacity/self-driving-car-sim