HIGH SCHOOL EDITION
Journal of Student Research

Volume 10 Issue 2 (2021)

Effects of Image Augmentation on Efficiency of a Convolutional Neural Network of a Self-Driving Car

Kiaan Upamanyu[1] and Indukala P.Ramaswamy[1]

[1]Greenwood High International School, Bangalore, Karnataka, India
DOI: https://doi.org/10.47611/jsrhs.v10i2.1486

ABSTRACT

The objective of this paper was to study the effectiveness of image augmentation techniques in training a Convolutional Neural Network (CNN) of a self-driving car and identify the most suitable form of image augmentation technique, using the Udacity Car Simulator. Firstly, a dataset of augmented and non-augmented images from a training track, consisting of left-, right-, and front-facing views from the car cameras was created. Various image augmentation techniques were used: zoom, brightness, pan, flip, random (augments the image by arbitrarily choosing a technique from the previous four), and no augmentation. Secondly, training datasets consisting of the aforementioned images and a log of car turning angles, throttle, and brake were built. The final training datasets were then used with NVIDIA training method to train different CNN. The different trained networks generated steering commands from the front-facing camera of the simulation and test track had no effect on the generalization of the CNN. Lastly, different trained networks were used on the test track of Udacity Car Simulator to calculate the following variables: distance travelled, and number of crashes made by the car. After these values were acquired, an efficiency analysis was performed. The results suggested augmentation of training data is a crucial factor when it comes to the process of generalizing a model to perform tasks. Random augmentations performed the best, but a combination of flip and brightness augmentations performed equally efficiently.

Introduction

Self-driving vehicles are automobiles in which human drivers are never required to take control to safely operate the vehicle. Also known as autonomous or "driverless" vehicles, they combine sensors and software to control, navigate, and drive the vehicle (Union of Concerned Scientists, 2017). There are six primary levels of autonomy in self-driving cars.

- Level 0: All systems are controlled by humans.
- Level 1: Cruise control or automatic braking may be controlled by the car; individuals are responsible for steering.
- Level 2: The car offers at least two simultaneous automated functions, like acceleration and steering, but requires humans for safe operation.
- Level 3: The car can handle all safety-critical functions, but the driver is required to be vigilant as he/she may have to take control when alerted.
- Level 4: The car is fully autonomous in some driving situations.
- Level 5: The car is adept to self-drive in all scenarios.

The basic concept in delivering autonomy involves mapping images or patterns to driving action. This is accomplished by three steps: image capture, deep learning using image sequences and deep learning directed response or action such as steering left or right or driving straight, accelerate or brake.

Pattern recognition has been revolutionized by Convolutional Neural Networks (CNN). Before the extensive adoption of CNN, pattern recognition tasks were performed using an initial stage of hand-crafted feature extraction followed by a classifier. CNN are used for their ability to recognize features from data automatically from an annotated input. The CNN approach is potent for image recognition tasks because the convolution operation captures the 2D nature of images. Also, by using the convolution kernels to scan an entire image, relatively few parameters need to be learned compared to the total number of operations.

A key determinant of the efficiency of a CNN is the quantity, quality and variability of images presented to it in the training dataset. Image augmentation is one method used to achieve this objective. Image augmentation increases the variety of a training data by creating modified versions of available images and thereby increasing the ability of the model to generalize. Various image augmentation techniques have been used in autonomous driving and include: (1) flip augmentation, (2) panned augmentation, (3) zoom augmentation, (4) brightness augmentation, (5) random augmentation and (6) no augmentation.

The objective of this paper was to study the effectiveness of image augmentation in training a CNN of a self-driving car and identify the most suitable form of image augmentation technique using the Udacity Car Simulator. The performance parameters chosen to study effectiveness were the percentage of the driving track covered by the car and the number of crashes. .

Methodology

For the purpose of this research a CNN was coded (Appendix-1) to have the car steer itself on the test track. The training data was captured from the training track. Pictures of the left-, right-, and forward-looking directions of the car were captured as the car went around the track. Simultaneously, the speed and the turning angle were also calculated. Before the CNN was trained, the images went through an augmentation process to increase the diversity of the training data and to expose the CNN to more scenarios. In this process, various kinds of augmentation were used. Following the training of the CNN, training and validation loss for each augmentation method was plotted. Finally, the model was connected to the Udacity Car Simulator to analyze the performance of the car using the code in Appendix-2. Table 1 highlights terms and definitions relevant to the study.

Table 1. Common terms and their definitions

Epochs	The number of times an algorithm goes through the entire training data set.
Training set	Set of data used in training a machine learning algorithm.
Validation set	Set of data used to evaluate how well a machine learning model has been trained
Training loss	The error on the training set of data.
Validation loss	The error on the validation set of data.

The various steps of training and validating the data are detailed below:

The Convolutional Neural Network (CNN)

Due to the high image and pattern recognition capabilities, a CNN was used for the self-driving car in the study. The CNN learned the method of steering a vehicle, along with recognizing the correct scenario for the appropriate steering angle. The CNN took a tremendously large amount of time to complete training itself. In this research, the neural network was trained for ten epochs and took 3 to 4 minutes per epoch. The model of the network is the Nvidia training model and is described in detail further on.

The Data

The training data was derived from the training tract on the simulator and consisted of training images and a log of car turning angles, throttle, and brake. With regards to training images, three sets of images: left-, right-, and front-facing point of views of the car, were acquired at each point of the track. Examples of training images captured from the three views are shown in Figures 1 to 3 respectively.

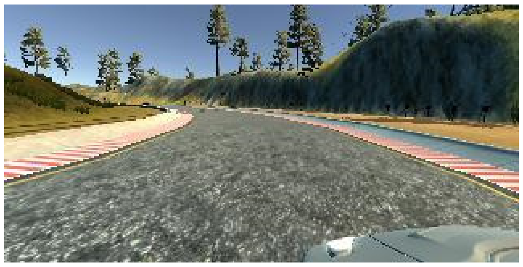


Figure 1. Left view of the car

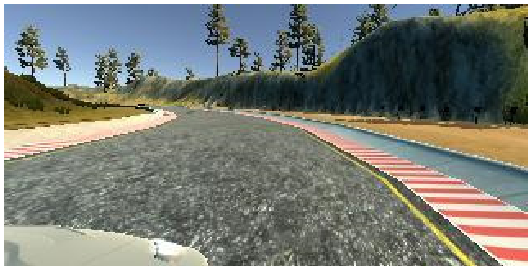


Figure 2. Right view of the car

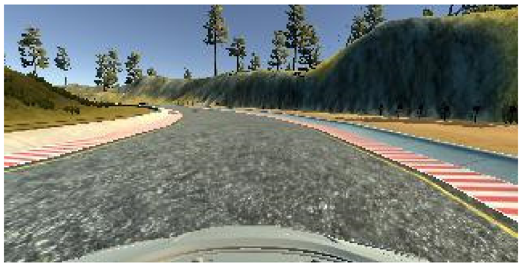


Figure 3. Center View of the car

There were 2 different training datasets each of different sizes. The first training dataset, *trainingdataset1*, had images from the car going around the track for three laps in an anticlockwise direction only. The second training dataset, *trainingdataset2*, had images from six laps around the track, three laps in the anticlockwise direction and three in the clockwise direction.

A ".csv" file was also used to correlate the turning angle of the car at a particular point to the images of the car's view at that point. All the files can be obtained on this GitHub repository (Upamanyu, 2020).

Table 2. Example of a row in the Driving_Log file

Center Image Location	Left Image Location	Right Image Location	Turning angle	Throttle	Breaking	Speed
C:\Users\Kiaan Upamanyu\One Drive\Desktop\research101\trainingdataset1\IMG\center_2020_11_28_19_36_06_537.jpg	C:\Users\Kiaan Upamanyu\One Drive\Desktop\research101\trainingdataset1\IMG\left_2020_11_28_19_36_06_537.jpg	C:\Users\Kiaan Upamanyu\One Drive\Desktop\research101\trainingdataset1\IMG\right_2020_11_28_19_36_06_537.jpg	0	0.731632	0	9.248178

If the car turns to the right, the turning angle is a positive number from 0 to 1; if the car turns to the left, the turning angle is a negative value from 0 to -1. If the car is not turning (driving straight), then the turning angle is 0. The car can attain a speed of any value between 0 to 30 (as there is no unit on the simulator) and has a maximum speed of 30.

Training of the Network

While training the network, a variety of factors were considered. These factors included a turning angle that was not biased to one side, the type of augmentation used on the training data, and the structure of the model used. The network was trained for ten epochs, and the data was randomly split into training and validation data. The training loss is calculated during an epoch while the validation loss is calculated when the epoch is completed. A lower validation loss would imply better performance of the car causing a greater travel distance and fewer to no crashes. A graph showing both the training and validation loss for each of the networks was also obtained to try and understand the effect of different augmentations and how it may cause overfitting, underfitting, or show that the dataset is an unrepresentative one. The image augmentations tested were flip, panned, zoom, brightness, no augmentation, random augmentation, and a combination of flip and brightness augmentation.

Flip Augmentation

In the flip augmentation, the image is flipped over the Y-axis, and the turning angle changes its sign from "+" to "-" and vice versa. Figure 4 is an example demonstrating a steering angle of 0.05154746 which becomes -0.05154746 in the altered image.

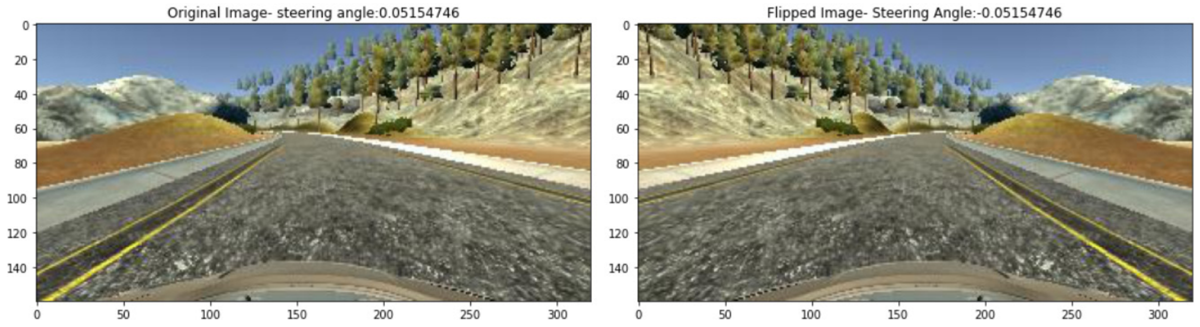

Original Image- steering angle:0.05154746
Flipped Image- Steering Angle:-0.05154746

Figure 4. Example of Flip Augmentation

Panned Augmentation

The panned augmentation is a kind of affine transformation. In the study, a translational shift of 10% towards the left and right were used as arguments for the Affine function(The code snippet: pan = iaa.Affine(translate percent = {"x" : (-0.1, 0.1), "y" : (-0.1, 0.1)}})). Figure 5 is the example of the panned augmentation.

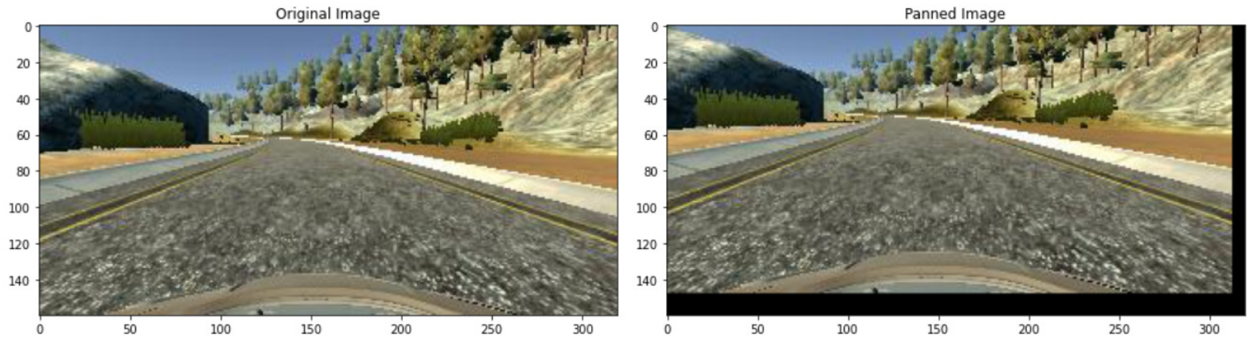


Figure 5. Example of Panned Augmentation

Zoom Augmentation

Zoom augmentation zooms or focuses on the original image. This results in new pixel values added around or pixel values incorporated in the altered image. Figure 6 is an example of zoom augmentation.

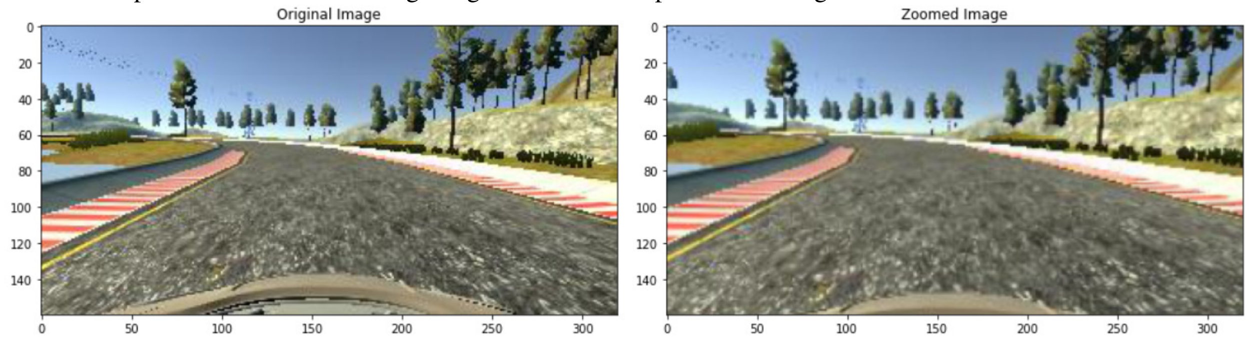


Figure 6. Example of Zoomed Augmentation

Brightness Augmentation

The brightness augmentation makes the original image brighter or dimmer randomly and exposes the network to the augmented image accordingly. Figure 7 is an example of the brightness augmentation where the altered image is darker.

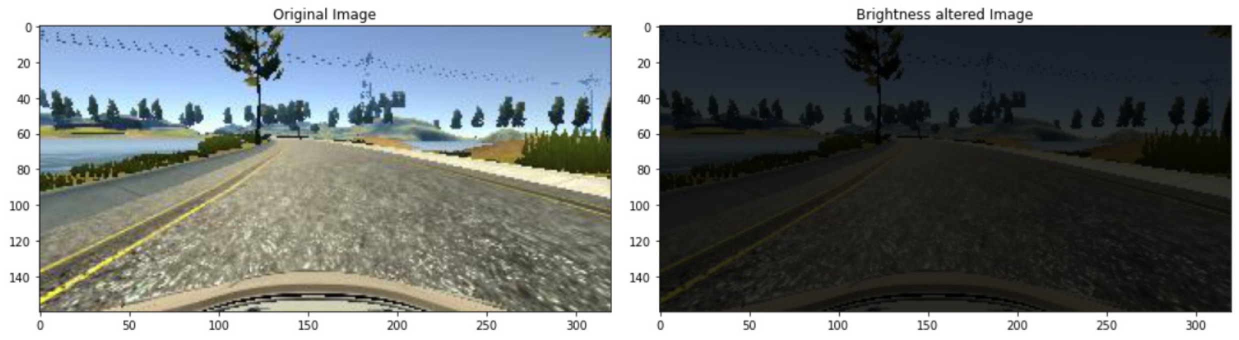


Figure 7. Example of Brightness Augmentation

Random Augmentation

In the network trained by randomly augmented training data, the four previously discussed techniques are implemented, and the network is exposed to all the augmentations and a variety of scenarios. The code snippet below allows the computer to randomly select an augmentation technique:

```
def random_augment(image, steering_angle):
  image = mpimg.imread(image)
  if np.random.rand() < 0.5:
    image = pan(image)
  if np.random.rand() < 0.5:
    image = zoom(image)
  if np.random.rand() < 0.5:
    image = img_random_brightness(image)
  if np.random.rand() < 0.5:
    image, steering_angle = img_random_flip(image, steering_angle)
  return image, steering_angle
```

No Augmentation

The network trained with no augmentation is trained solely on the original images captured by the Udacity simulator. Following the augmentations, the images are preprocessed to enhance the image features by suppressing undesirable distortions. Unnecessary features like the hood of the car and portions of the image that do not contain the road are cropped. In figure 8, the original image is cropped from values 0-60 and from 135-160 on the Y-axis. Besides cropping, certain modifications were implemented to enhance the image. In the Nvidia model, YUV images were inputted into the network. Hence, a YUV color-coding system was also used in this study. A Gaussian blur was used to reduce noise and smoothen the image. The size of the image then reduced 66 by 200 to fit the input size in the Nvidia model as can be seen in the preprocessed image in figure 8.

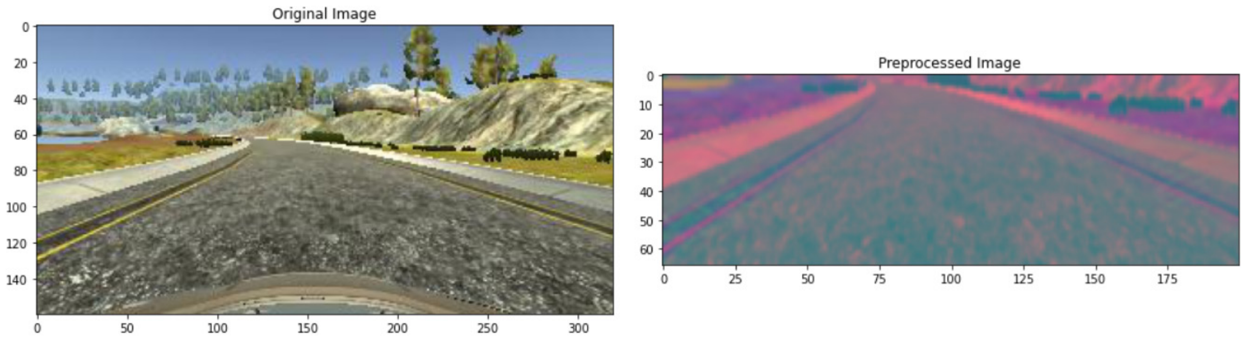


Figure 8. Original & Preprocessed Image

The Training Model: Nvidia model

The model used here was the Nvidia model, a useful model used for behavioral cloning. The architecture of the model was obtained from the publication “End to End Learning for Self-Driving Car” (Bojarski, et al., 2016).

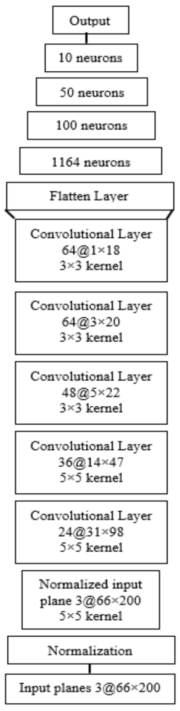

Output
10 neurons
50 neurons
100 neurons
1164 neurons
Flatten Layer
Convolutional Layer 64@1×18 3×3 kernel
Convolutional Layer 64@3×20 3×3 kernel
Convolutional Layer 48@5×22 3×3 kernel
Convolutional Layer 36@14×47 5×5 kernel
Convolutional Layer 24@31×98 5×5 kernel
Normalized input plane 3@66×200 5×5 kernel
Normalization
Input planes 3@66×200

Figure 9. CNN architecture

In defining the model, normalization was not required as the data was already normalized. The normalized data passes into the first convolutional layer. As can be seen in Figure 9, the first convolutional layer has 24 filters and a kernel size of 5x5. The second convolutional layer has 36 filters and a kernel size of 5x5. The third layer has 48 filters and a kernel size of 3x3. The fourth and fifth layers are identical and have 64 layers and kernel sizes of 3x3. The subsample refers to the stride length of the kernel as it processes through the image. A stride length of 2 by 2 was used for the first three layers to increase the computation speed, followed by the 'elu' (exponential linear unit) activation function. The fourth and fifth layers do not require any pixels to be skipped, so a stride length of 1 pixel was used. The flatten layer receives the output from the previous convolutional layer and converts it into a single 1-dimensional array. Following the flatten layer, four dense layers are added with 100, 50,10, and 1. The first three have the same activation function as 'elu', and the final layer outputs the predicted steering angle for the self-driving car.

Testing and hypothesis

Before testing was conducted, an assumption was made that as the network would be exposed to a more extensive variety of training data when the training data had undergone random augmentation, the performance would be the best and relatively similar to a network trained by training data which had undergone a combination of brightness and flip augmentations. Brightness and Flip augmentations were chosen as the network would be exposed to a lot of situations which involve shadows and a vast variety of turns varying from steep to gradual and left to right turns on the test track. There is no data captured while the car is driving on the test track, so the results do not affect the network.

Results and Discussion

The results obtained after applying the model to Udacity Car Simulator are presented, analyzed, and discussed in this section. The two datasets - *traningdataset1* and *trainingdataset2* - were generated for training purposes. Histograms for each dataset were obtained and evaluated for distribution and bias as discussed below.

The following graphs are for *trainingdataset1* (Figures 10 to 12)

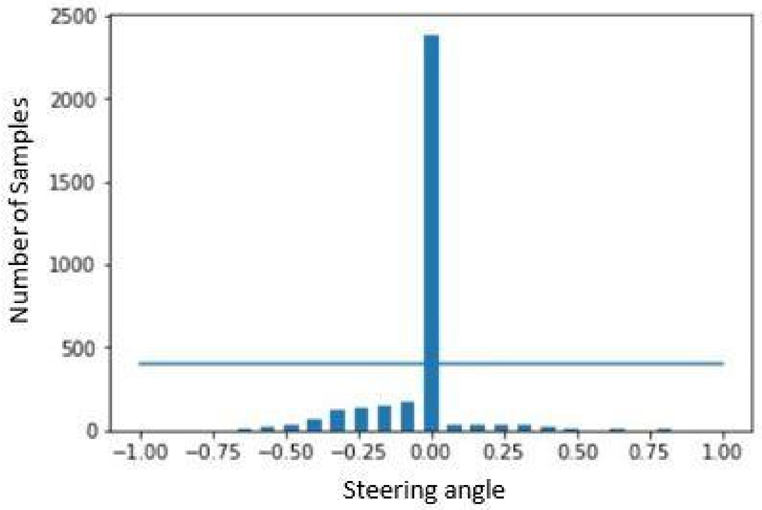


Figure 10. Histogram of training data before samples are removed in trainingdatset1

The graph in Figure10 has an X-axis representing the steering angle from -1 to 1, and the number of samples is shown on the Y-axis. Each sample represents a particular steering angle. As seen in the figure, there are a high number of samples for the 0-steering angle, that trains the car with a bias to drive straight. All the samples below the blue horizontal line represent a uniform distribution. So, if the samples above the blue horizontal line are removed, the model will not be heavily skewed to always predict the 0 angle. After removing the samples, the following histogram was deduced (Figure 11).

```
total data: 3284
removed 1989
remaining: 1295
[<matplotlib.lines.Line2D at 0x7f7fd41718d0>]
```

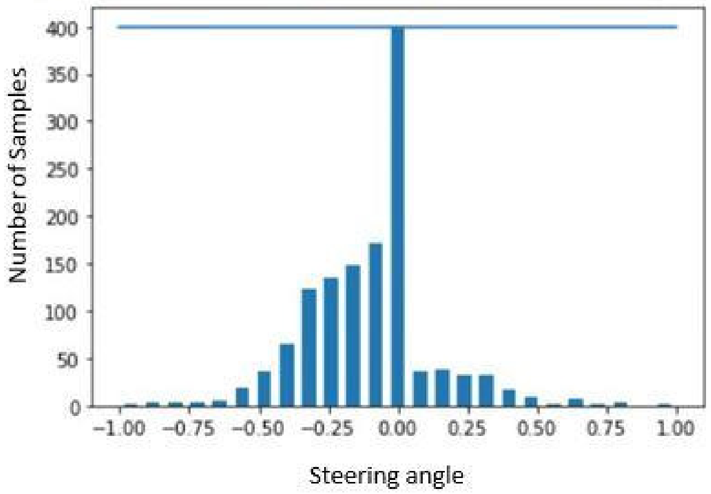


Figure 11. Histogram of training data after samples are removed from *tainingdataset1* (1989 samples removed)

As can be seen from Figure 11, 1989 samples were removed from the training set. But there is a slight bias to turning left because of the high number of left turns present in the training track. In *trainingdataset1* the car is made to go around the training track only in the anticlockwise direction, affecting the data.

The training data was then randomly split by the code into two sets: training and validation sets. The histograms of the two types of sets are shown in Figure 12.

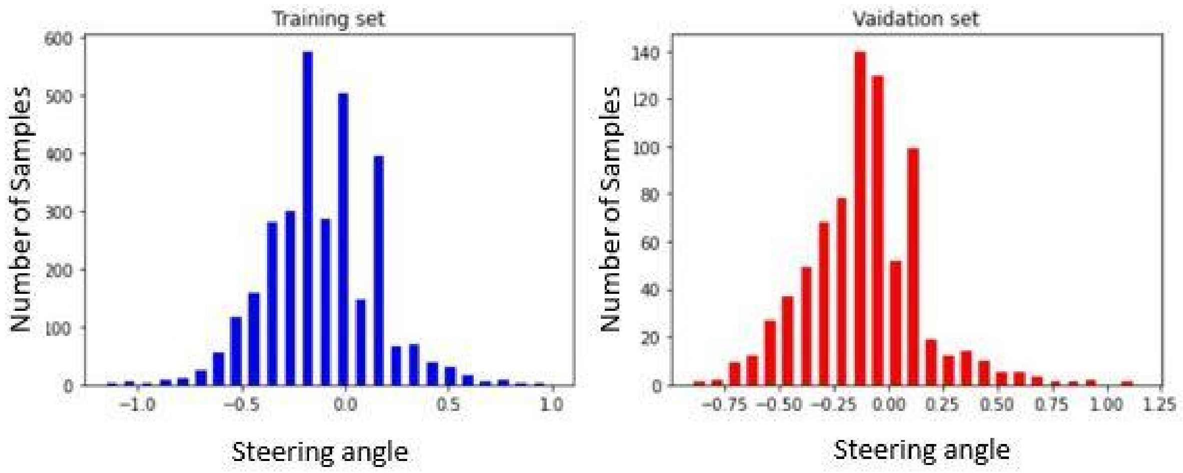


Figure 12. Histogram of training & validation sets for *trainingdataset1*

Both the histograms follow a general trend, but there was still a bias towards turning left. This process was then repeated for the *trainingdataset2*, and the following histograms were obtained (Figure 13 to 15).

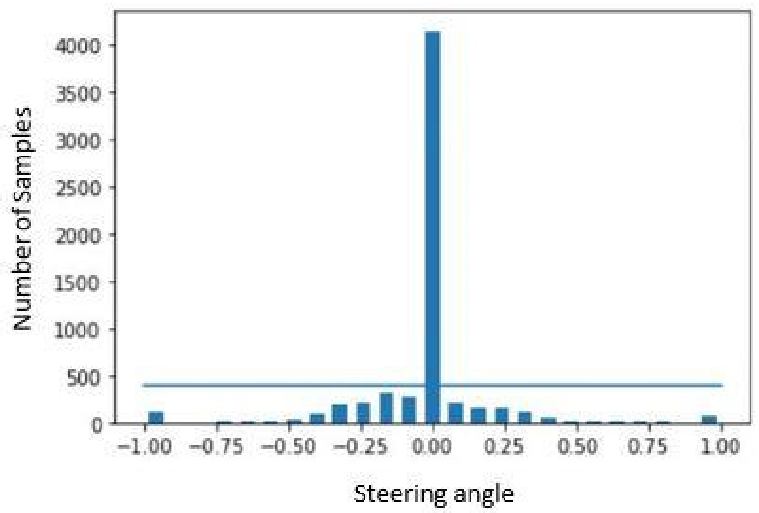


Figure 13. Histogram of training data before samples are removed for *trainingdataset2*

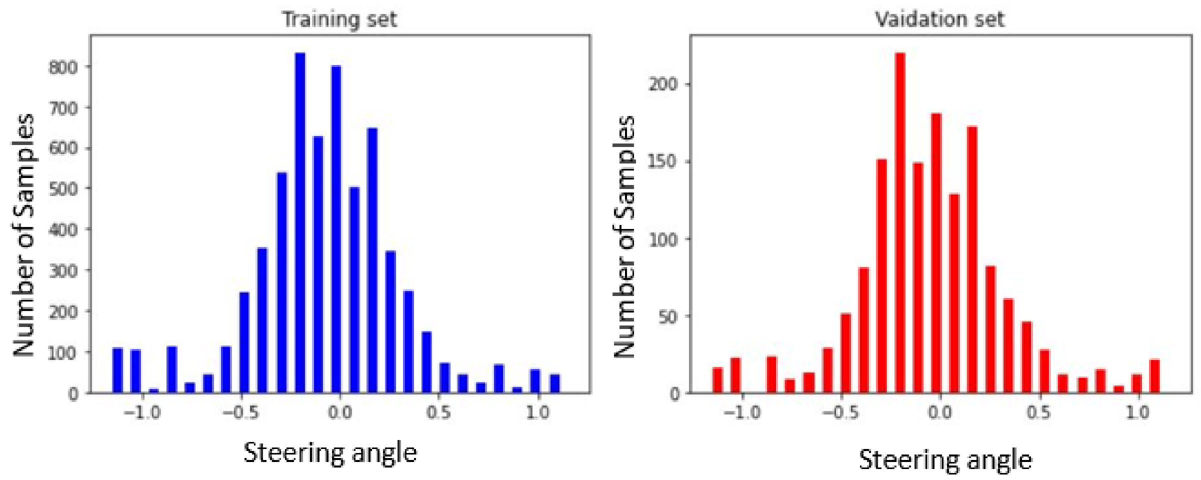
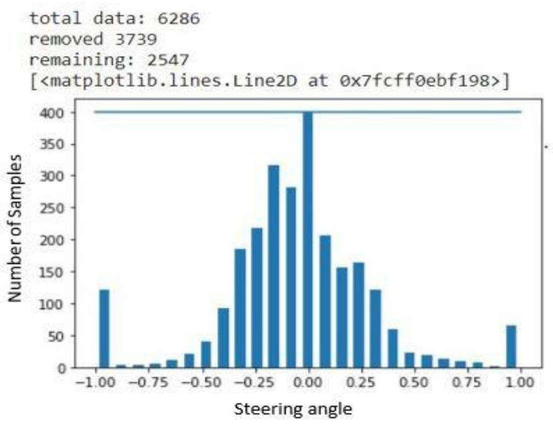


Figure 14. Histogram of training data after samples are removed from *trainingdataset2* (3739 samples removed)



```
total data: 6286
removed 3739
remaining: 2547
[<matplotlib.lines.Line2D at 0x7fcff0ebf198>]
```

Figure 15. Histogram of training and validation sets for *trainingdataset2*

As can be seen from Figure 13, 14, and 15, the data is much more evenly distributed, and the training and validation sets are also following a remarkably similar trend to each other. Therefore, *trainingdataset2* which involved six laps, with three laps in clockwise and anticlockwise direction each was better than *trainingdataset1*, which was derived from only three laps in the anticlockwise direction.

During testing, the car was allowed 3 attempts and the following performance parameters were evaluated: the percentage of driving tack covered by the car, the number of crashes and whether the car was able to get back on road. If the car was unable to begin driving after 5 seconds, the test was stopped.

Models Trained with No Augmentation

The graphs in Figure 16 to 22 represent the training and validation loss for each model training with various augmentation methods (Blue line represent training loss and yellow line represents validation loss). The model is overfitting if the training loss is lesser than the validation loss, underfitting if the training loss is greater than the validation loss and is perfect if the training loss is similar to validation loss. The first augmentation tested is no augmentation and Figure 16 shows the graphs for no augmentation for both training datasets.

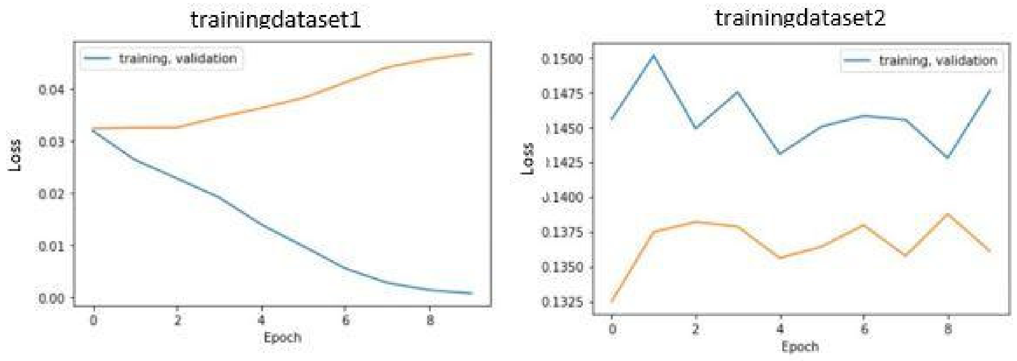


Figure 16. No Augmentation Loss Graph for *trainingdataset1* & *trainingdataset2*

Both the models performed exceptionally poorly. In *trainingdataset1*, the car travelled an awfully short distance before turning entirely to the left and crashing. All three tests provided the same result. In the *trainingdataset2* model, the car went a bit further before having the same result and crashing. In both instances, the car did not try to correct itself to the right path.

Models Trained with Brightness Augmentation

The next models tested had data augmented by the brightness augmentation. The following were the graphs.

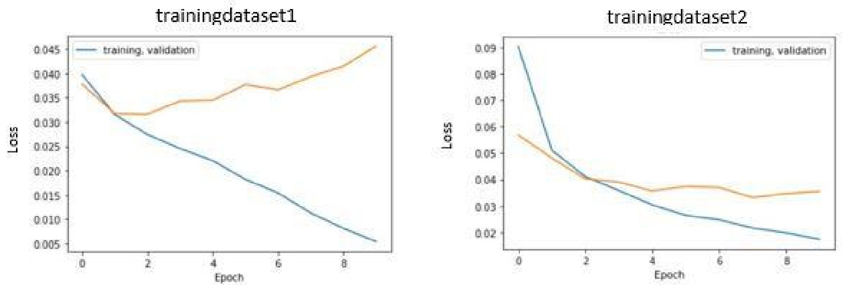


Figure 17. Brightness Augmentation Loss Graph for *trainingdataset1* & *trainingdataset2*

The model performed poorly here as well but comparatively better to no augmentation. Both training dataset models made it past the first turn but crashed shortly after. The car was able to self-correct itself a few times but failed to turn right and escape the collision.

Models Trained with Flip Augmentation

The next augmentation tested was the flip augmentation. Reflecting on the previous tests, it was assumed that this augmentation would provide much better results as it provides the samples to allow generalization for steering the vehicle. Figure18 represents the loss graphs.

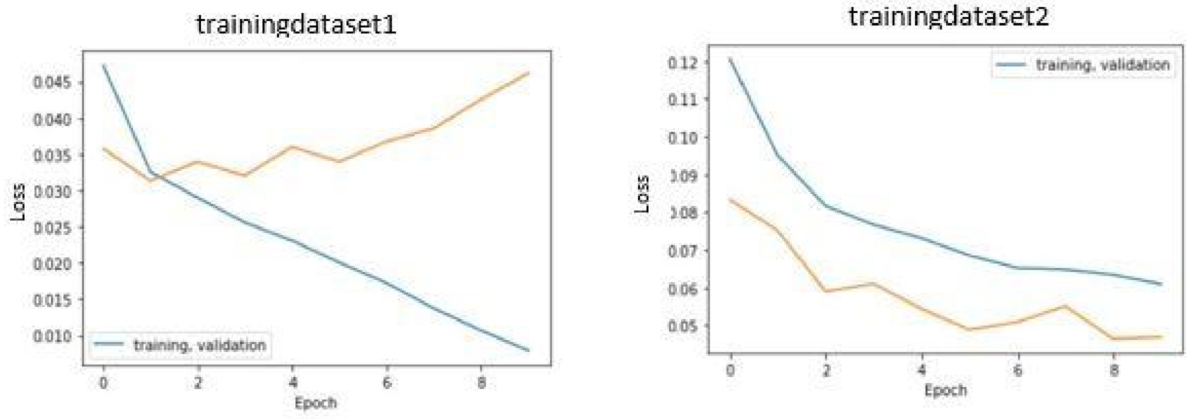


Figure 18. Flip Augmentation Loss Graph for *trainingdataset1* & *trainingdataset2*

While the graph for *trainingdataset1* does not show a similar trend, the graph for *trainingdataset2* shows a validation curve with a lower validation loss and a much better trend to the training loss. As expected, *trainingdataset1's* model performed poorly and crashed very quickly and did not show any scope of correcting its course. But *trainingdataset2's* model performed much better and was able to correct itself after crashing once but crashed again on the right side shortly after. This result demonstrated that the original assumption that the brightness and flip augmentation combined is the ideal augmentation combination could be valid and this is discussed in more depth further in the paper.

Models Trained with Pan Augmentation

The next augmentation is the pan augmentation. Figure 19 shows the graph for this augmentation.

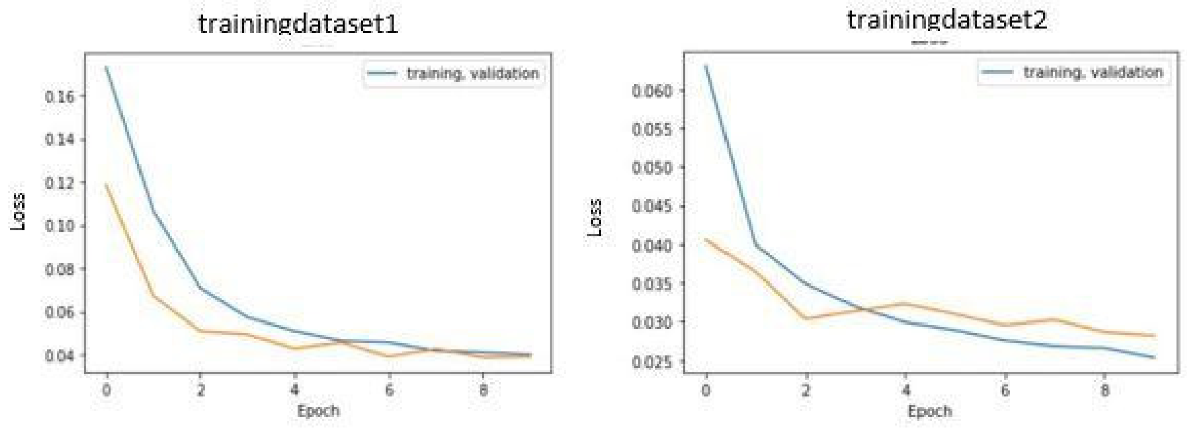


Figure 19. Pan Augmentation Loss Graph for *trainingdataset1* & *trainingdataset2*

The model trained with *trainingdataset1* and panned augmentation crashed early in all the three trials and did not rectify its steering. But the model trained with *trainingdataset2* and panned augmentation performed as good as the model trained with *trainingdataset2* and flip augmentation. The performance results of panned and flip augmentation individually suggest that a combination of panned and flip augmentation merits further testing in future studies.

Models Trained with Zoom Augmentation

The zoom augmentation was the next augmentation tested. Figure 20 shows the loss graphs obtained for the models.

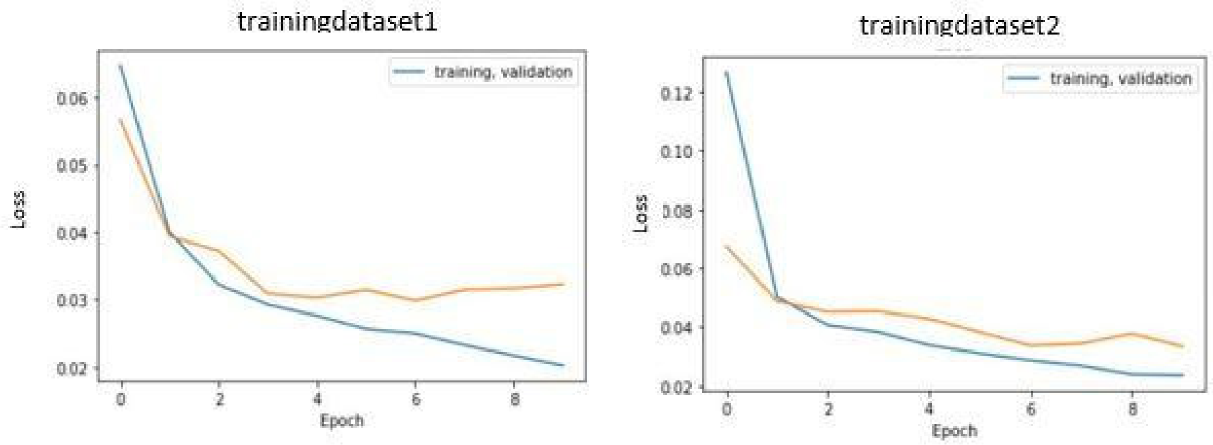


Figure 20. Zoom Augmentation Loss Graph for *trainingdataset1* & *trainingdataset2*

Both the models did not perform well, even though the validation and training curves for both models were relatively similar. Both models caused two crashes at the incredibly early stages of the test track, and this was the case for all three tests. The observations made from all the previous tests show that individual augmentations performed poorly. Consequently, it may be inferred that the model performs better when there is a combination of augmentations.

Models Trained with Random Augmentation

The following were the loss graphs obtained for random augmentation.

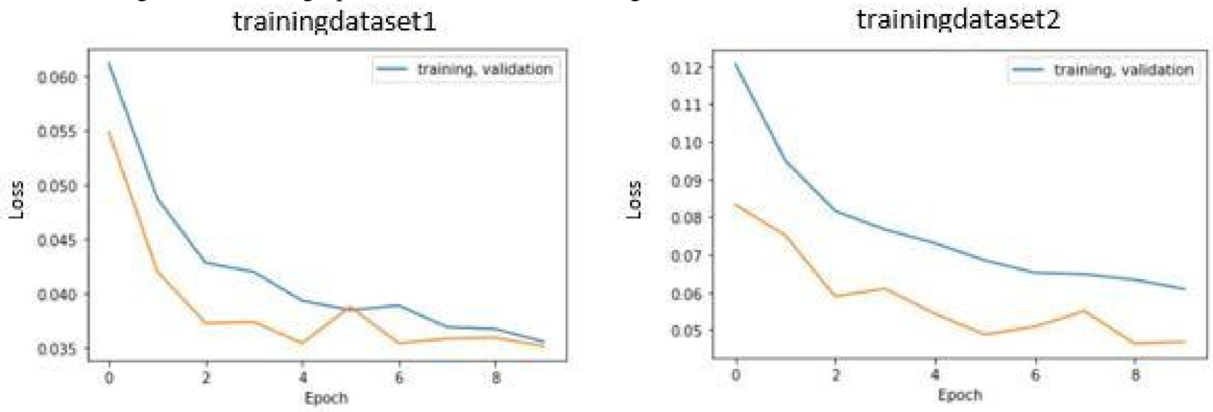


Figure 21. Random Augmentation Loss Graph for *trainingdataset1* & *trainingdataset2*

As hypothesized, both the random augmentations performed exceptionally well, with the model trained by *training-dataset2* outperforming the model trained by *trainingdataset1*. Both models were able to rectify their turning angle.

Sometimes they turned with only a small margin of error before crashing. All three tests were successful, and the cars were able to cover almost 70-80% of the track without a single crash.

Models Trained with Flip and Brightness Augmentation

The final augmentation combination tested was of the assumption made before testing a combination of brightness and flip augmentation. The following were the loss graphs obtained.

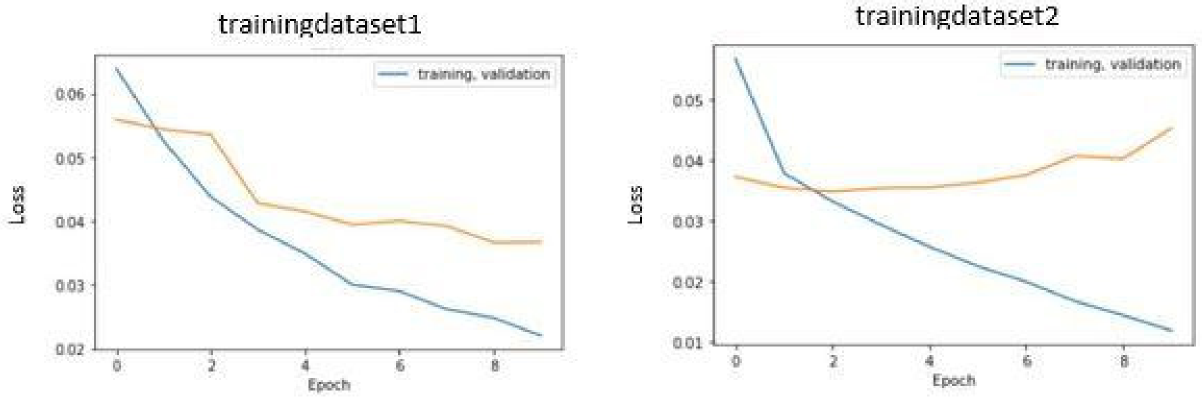


Figure 22. Flip and Brightness Augmentation Loss Graph for *trainingdataset1* & *trainingdataset2*

While the validation loss curve does not look similar to the training loss curve, the car drove unexpectedly well and covered nearly 60-75% of the track in its tests. The flip and brightness combined augmented training data model performed exceedingly well and was quite similar to the randomly augmented data model. This demonstrates that while the model works extremely well with random augmentation, the CNN definitively works well with a combination of the brightness and flip augmentation as shown by the results in Table 3 and Table 4. An exception to this statement would be the performance of the model trained with panned augmented data. The pan augmentation technique despite being trivial in merely putting the image in a panel, still performed remarkably similar to the model trained with flip augmented data which is counter intuitive.

Post analysis of the validation and training loss graphs, two graphs were extremely peculiar, the graph of trainingdataset2 with No augmentation and Random augmentation (Figure 16 and Figure 21). This was a surprising development in the study and on further review two possible reasons could have caused this. The first being that Regularization was not applied, which may have caused the validation loss to be lower. Regularization is a method used for adjusting the function by adding an extra penalty term to the error function. The second and less likely reason is that the validation set may have been simpler than the training set. This is unlikely as the training track is the same for both the training and validation sets. Further study must be done for a definitive answer regarding as to why the validation loss was lower than the training loss.

Table 3 and Table 4 summarize the study by tabulating the efficiency results of the self-driving car with the different augmentations. It compares the distance travelled by the car on the track and the number of crashes made by the car and if the car continued to travel after the crashes. Since the simulator does not actually measure the distance, the car travels a rough estimate in percentage of the distance travelled by the car was calculated through observation for the analysis.

Table 3. Car efficiency with *trainingdataset1*

Type of augmentation	Percentage of track covered	Number of crashes	Path rectification of car after crash
No Augmentation	5%	1	No

Brightness Augmentation	5%	1	No
Flip Augmentation	15%	1	No
Pan Augmentation	2%	1	No
Zoom Augmentation	10%	2	1st Crash: Yes 2nd Crash: No
Random Augmentation	40%	1	No
Flip and Brightness Augmentation	35%	1	No

Table 4. Car efficiency with *trainingdataset2*

Type of augmentation	**Percentage of track covered**	**Number of crashes**	**Path rectification of car after crash**
No Augmentation	10%	1	No
Brightness Augmentation	15%	1	No
Flip Augmentation	20%	1	No
Pan Augmentation	20%	1	No
Zoom Augmentation	15%	1	No
Random Augmentation	80%	1	No
Flip and Brightness Augmentation	75%	1	No

Conclusion

Self-driving cars are developing at an extremely rapid rate, and the neural networks running them are becoming even more efficient. After the testing of the coded convolutional neural network using the Udacity car simulator, it can be deduced that augmentation of training data is a crucial factor when it comes to the process of generalizing a model to perform tasks. While exposing a network to a wide variety of situations is extremely important for generalization, not all augmentations have the same effect in the case of the self-driving car network model used here. The most necessary augmentations were the brightness and flip augmentation, but when used individually, do not help in training a generalization model of a self-driving car. When used in combination with each other and with a large dataset of images, the generalization process becomes extremely efficient.

Further Study

The self-driving car model tested here was simple but extremely useful in testing the most efficient types of augmentations. Testing in a different simulator, like Microsoft's AirSim, will help understand what other augmentations will be required as the car would be exposed to more external circumstances, such as other vehicles, trees, zebra crossings, pedestrians, and more road lanes. In such an environment, the model requires to be trained with a much larger set of training samples, but it is still plausible that the type of augmentation necessary for efficient performance is a combination of brightness and flip augmentations.

Acknowledgments

I would like to express my gratitude to my research supervisor and teacher, Ms. Indukala P. R. without whose guidance and support this research article would not have been possible. I would also like to thank Mr. Rayan Slim, the creator of the Udemy course on Self-driving cars. His course taught me a lot about coding automated cars and inspired me to do an in-depth study about image augmentation.

References

Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., . . . Zieba, K. (2016). *End to End Learning for Self-Driving Cars*. Nvidia. Retrieved from https://arxiv.org/abs/1604.07316

Brownlee, J. (2019, February 27). *Machine Learning Mastery*. Retrieved from How to use Learning Curves to Diagnose Machine Learning Model Performance: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Canuma, P. (2018, October 11). *Towards Data Science*. Retrieved from Image Pre-processing: https://towardsdatascience.com/image-pre-processing-c1aec0be3edf

Keras. (n.d.). *Layer activation functions*. Retrieved from Keras: https://keras.io/api/layers/activations/

Slim , R., Sharaf, A., Slim, J., & Tanveer, S. (2020, September). The Complete Self-Driving Car Course - Applied Deep Learning. *The Complete Self-Driving Car Course - Applied Deep Learning*. Udemy. Retrieved from https://www.udemy.com/course/applied-deep-learningtm-the-complete-self-driving-car-course/

Slim, R. (2020, May 12). *Self-Driving-Car-Course-Codes/Section 14 - Behavioral Cloning/*. Retrieved from Github: https://github.com/rslim087a/Self-Driving-Car-Course-Codes/tree/master/Section%2014%20-%20Behavioral%20Cloning

Udacity. (2019, January 11). *udacity/self-driving-car-sim*. Retrieved from Github: https://github.com/udacity/self-driving-car-sim

Union of Concerned Scientists. (2017, January 26). *Union of Concerned Scientists*. Retrieved from Self-Driving Cars Explained: https://www.ucsusa.org/resources/self-driving-cars-101

Upamanyu, K. (2020, December 27). *Kiaan1204 / TRAINING-DATA-1-AND-2*. Retrieved from Github: https://github.com/Kiaan1204/TRAINING-DATA-1-AND-2

Wang, J., & Perez, L. (2017, December 13). *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. Retrieved from arXiv: https://arxiv.org/abs/1712.04621

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2020). *Dive into Deep Learning*. Retrieved from https://d2l.ai/chapter_computer-vision/image-augmentation.html