DEEP LEARNING PROJECT REPORT

End-to-end Learning for Self-Driving Cars

Instructor: Dr. M. Daud Abdullah Asif

Submitted by: M. Hasnain Naeem (Group 1, 212728)
mnaeem.bscs17seecs@seecs.edu.pk

1 ABSTRACT

In this project, the research paper titled “end-to-end learning for self-driving cars” [1] by Nvidia team has been reproduced. It involves training convolutional neural network (CNN) to predict the steering angle for each incoming video frame from the front-camera of the self-driving car. An end-to-end learning approach is used which requires no annotation and labeling of the objects in each frame. The approach shows promising results in terms of predicting accurate steering angle as well as in terms of model size and complexity. Other approaches employ multiple techniques, such as object segmentation, lane detection & marking, path planning, etc. to steer the car. But the employed technique simply requires training of model based off given video frames and the steering angle. Various Python packages are used for the implementation, for instance, OpenCV is used to read and vectorize video frames, TensorFlow is used to implement the deep learning model, and Gooey is to create the GUI for training and testing scripts. Lastly, some improvements, such as the usage of two-stream CNN along with LSTMs, have been suggested to improve the results.

2 INTRODUCTION

The popularity of CNNs skyrocketed for various reasons, major being reason being that it eliminated the need of extensive preprocessing of data for feature extraction using hand-crafted methods. CNNs are revolutionary in that features are automatically learned from training examples. Because the convolution process involved in CNNs captures the 2D aspect of images, the CNN technique is very effective in image identification applications. Furthermore, huge labeled datasets are available for training and validation of various CNN models. Availability of huge datasets have not only enabled researchers to achieve near human-level accuracy but CNNs are outperforming humans in certain use cases. Plus, with the recent advances in the realm of computer hardware, powerful GPUs have become accessible allowing CNN learning methods to be implemented with great learning and inference speed.

In this project, an unconventional usage of CNNs has been made, the CNN learns the complete processing chain required to control a car. The groundwork for this research was in a DARPA seedling project called the DARPA Autonomous Vehicle (DAVE), in which a subscale radio-controlled (RC) car was driven down a garbage-filled alley.

DAVE received hours of human driving experience for its training. Later, NVIDIA started a new initiative to expand DAVE and develop a reliable system for driving on public roads. The main goal of this research was to avoid the data collection and preprocessing steps such as having to recognize certain human-determined things like lane markings, crash barriers or other cars and to develop a set of if-then-else rules based on the observation of these features.

In this paper, we discuss the methodology employed, the preliminary results, and the implementation details for an end-to-end learning experiment along with the future direction.

3 IMPLEMENTATION

3.1 APPROACH

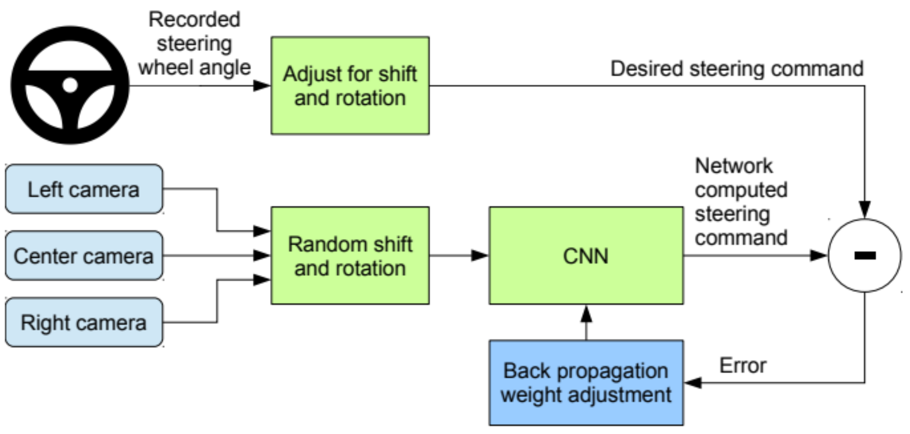

Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
Back propagation weight adjustment
Error

Figure 1: Block Diagram of the End-to-end Learning Approach for Training

The figure shows a block diagram of the training system used by Nvidia team. Images are sent to a CNN, which calculates a suggested steering command. The suggested command for the image is compared to the desired command and the CNN weights are changed to bring the CNN output closer to the desired output.

The difference between the proposed model and the reproduced model is that only center image is used to train the model. Left and right images could also have been used but only center images are used due to limited resources for training.

3.2 NETWORK ARCHITECTURE

The CNN architecture of the model used for training is shown in the figure below.

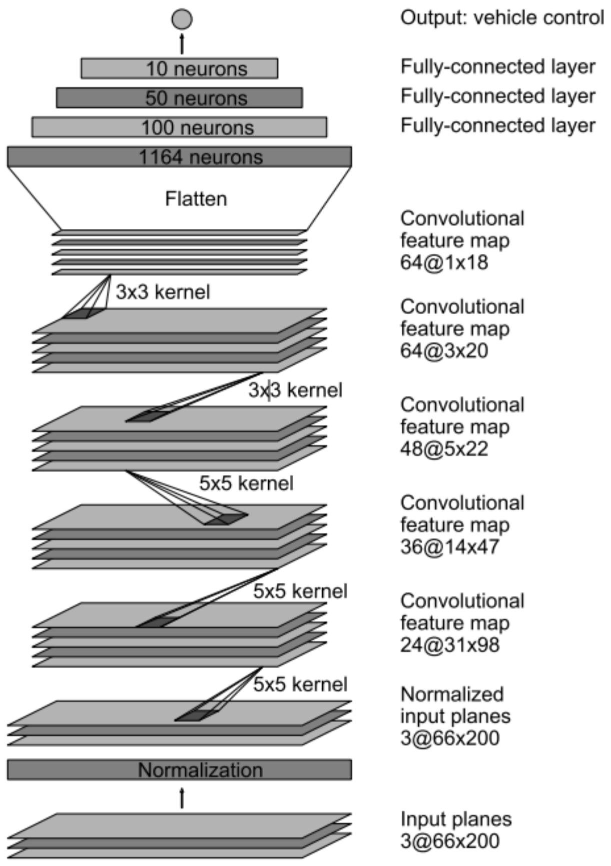


Figure 2: Implemented CNN Model Architecture

The convolutional layers used in the model were chosen empirically after iterating through various layer configurations. 5 x 5 kernel is used in the first 3 convolutional layers with 2 x 2 stride. 3 x 3 kernel with 1 x 1 stride is used in the last 2 convolutional layers. Finally, 3 fully connected layers are used at the end of convolutional layers.

Model architecture is pretty simple compared to other self-driving models which combine multiple approaches. Also, the model is shallow compared to the cotemporary CNN models like AlexNet, VGG, and GoogleNet.

3.3 Hyper-parameters

Epochs	30
Batch Size	100
Optimizer	Adam
L2_norm_const	0. 001

3.4 Tech Stack

Language: Python 3.x

Frameworks: TensorFlow (1.x or 2.x)/TensorFlow GPU (1.x or 2.x), OpenCV, NumPy, Gooey

3.5 Project Structure

- **Folders**
    - **driving_dataset**
        - **images:** default folder for training images.
        - **videos:** default folder for test videos.
        - **data.txt:** contains image names and corresponding steering angles.
    - **logs:** contains logs to be plotted and visualized in the TensorBoard.
    - **save:** contains model checkpoints.
    - **assets:** contains assets used by the software such as the steering wheel image.
- **Scripts**
    - **model.py:** code for CNN model.
    - **run.py:** script to test the model on real-time video feed.
    - **run_dataset.py:** script to test the model on images in the driving_dataset/images folder.
    - **run_video:** script to run the model on a video.
    - **train.py:** script to train the model.

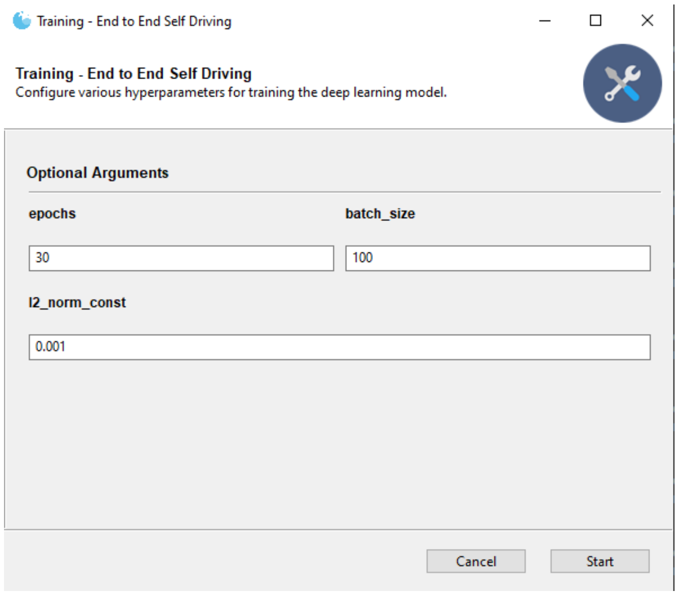
3.6 Usage

3.6.1 Training

1. Run following command in the project directory to train the model:

   python train.py

Running this will pop up a window shown in figure 3. All the hyper-parameters can be tuned from that window. Default values in the window gave best results for all the iterations performed using different parameters.



Training - End to End Self Driving

Training - End to End Self Driving
Configure various hyperparameters for training the deep learning model.

Optional Arguments

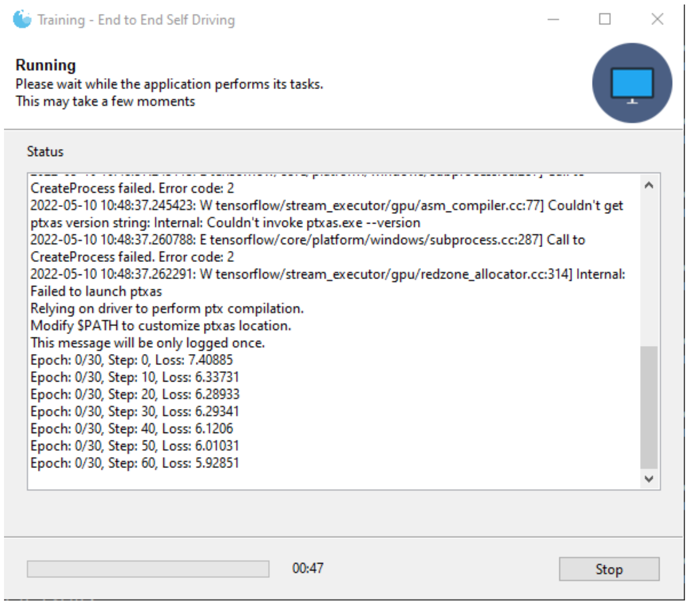
epochs: 30

batch_size: 100

l2_norm_const: 0.001

Figure 3: GUI for Training Settings

2. Press “Start” key to start training. That will display the window shown in figure 4. This window contains the logs about current epoch and loss.



Training - End to End Self Driving

Running
Please wait while the application performs its tasks.
This may take a few moments

Status

```
CreateProcess failed. Error code: 2
2022-05-10 10:48:37.245423: W tensorflow/stream_executor/gpu/asm_compiler.cc:77] Couldn't get
ptxas version string: Internal: Couldn't invoke ptxas.exe --version
2022-05-10 10:48:37.260788: E tensorflow/core/platform/windows/subprocess.cc:287] Call to
CreateProcess failed. Error code: 2
2022-05-10 10:48:37.262291: W tensorflow/stream_executor/gpu/redzone_allocator.cc:314] Internal:
Failed to launch ptxas
Relying on driver to perform ptx compilation.
Modify $PATH to customize ptxas location.
This message will be only logged once.
Epoch: 0/30, Step: 0, Loss: 7.40885
Epoch: 0/30, Step: 10, Loss: 6.33731
Epoch: 0/30, Step: 20, Loss: 6.28933
Epoch: 0/30, Step: 30, Loss: 6.29341
Epoch: 0/30, Step: 40, Loss: 6.1206
Epoch: 0/30, Step: 50, Loss: 6.01031
Epoch: 0/30, Step: 60, Loss: 5.92851
```

Figure 4: Training logs

3. (Optional) To stop the training at some point, press “Stop” key. The model with minimum loss will automatically be stored during the training process.

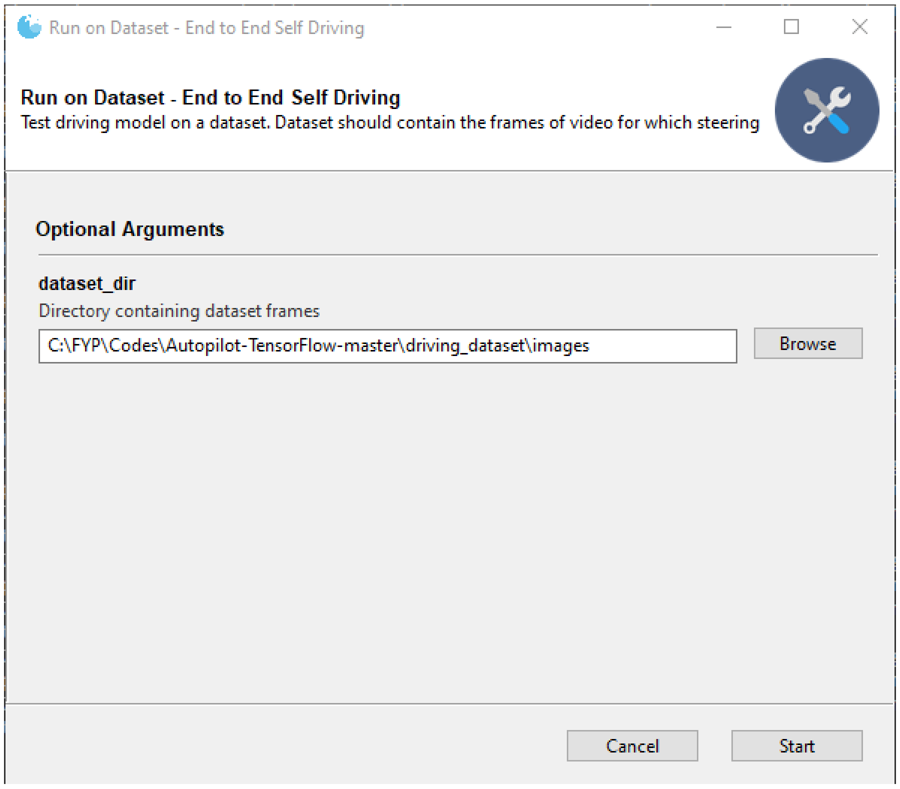
3.6.2 Testing

3.6.2.1 Testing on Image Dataset

Run following command in the project directory to test the model on image dataset:

Python run_dataset.py

It will open a window, as shown in figure 5, to select the directory containing the image dataset for testing the model.


Run on Dataset - End to End Self Driving

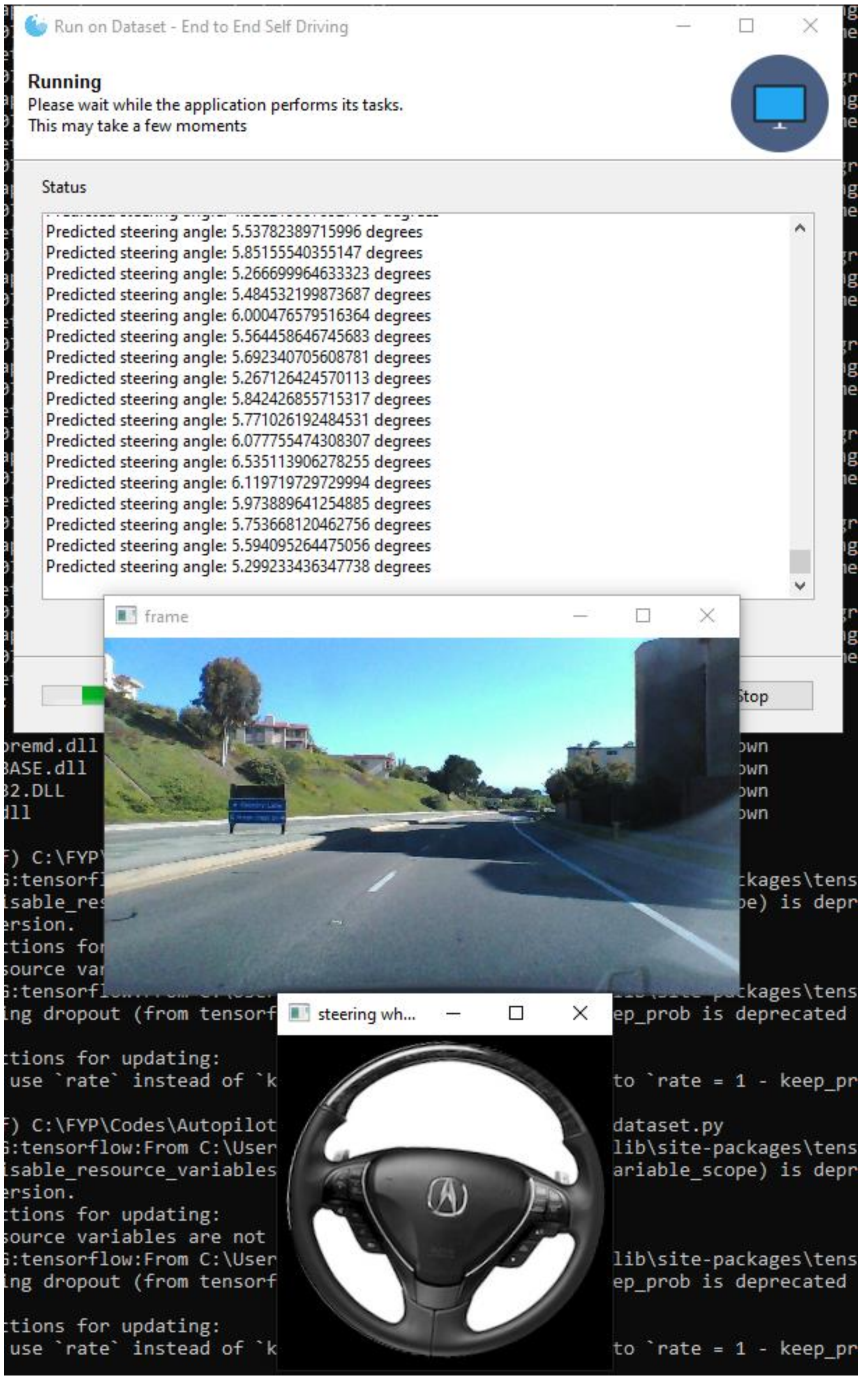
Run on Dataset - End to End Self Driving
Test driving model on a dataset. Dataset should contain the frames of video for which steering

Optional Arguments

dataset_dir
Directory containing dataset frames
C:\FYP\Codes\Autopilot-TensorFlow-master\driving_dataset\images

Figure 5: Image Dataset Folder Selection for Testing

After pressing “Start” button, the model will start predicting steering angle for each frame in the image dataset directory. Steering angle will be visualized using a steering wheel as well as it its numerical value will be predicted in the logs window, as shown in figure 6.



```
Running
Please wait while the application performs its tasks.
This may take a few moments

Status
Predicted steering angle: 5.53782389715996 degrees
Predicted steering angle: 5.85155540355147 degrees
Predicted steering angle: 5.266699964633323 degrees
Predicted steering angle: 5.484532199873687 degrees
Predicted steering angle: 6.000476579516364 degrees
Predicted steering angle: 5.564458646745683 degrees
Predicted steering angle: 5.692340705608781 degrees
Predicted steering angle: 5.267126424570113 degrees
Predicted steering angle: 5.842426855715317 degrees
Predicted steering angle: 5.771026192484531 degrees
Predicted steering angle: 6.077755474308307 degrees
Predicted steering angle: 6.535113906278255 degrees
Predicted steering angle: 6.119719729729994 degrees
Predicted steering angle: 5.973889641254885 degrees
Predicted steering angle: 5.753668120462756 degrees
Predicted steering angle: 5.594095264475056 degrees
Predicted steering angle: 5.299233436347738 degrees
```

```
premd.dll                                        own
BASE.dll                                         own
32.DLL                                           own
dll                                              own

F) C:\FYP\
6:tensorf                                        kages\tens
isable_res                                       pe) is depr
ersion.
tions for
source var
6:tensorfl                                       ite-packages\tens
ing dropout (from tensorf                        ep_prob is deprecated

tions for updating:
 use `rate` instead of `k                        to `rate = 1 - keep_pr

F) C:\FYP\Codes\Autopilot                        dataset.py
6:tensorflow:From C:\User                        lib\site-packages\tens
isable_resource_variables                        ariable_scope) is depr
ersion.
tions for updating:
source variables are not
6:tensorflow:From C:\User                        lib\site-packages\tens
ing dropout (from tensorf                        ep_prob is deprecated

tions for updating:
 use `rate` instead of `k                        to `rate = 1 - keep_pr
```

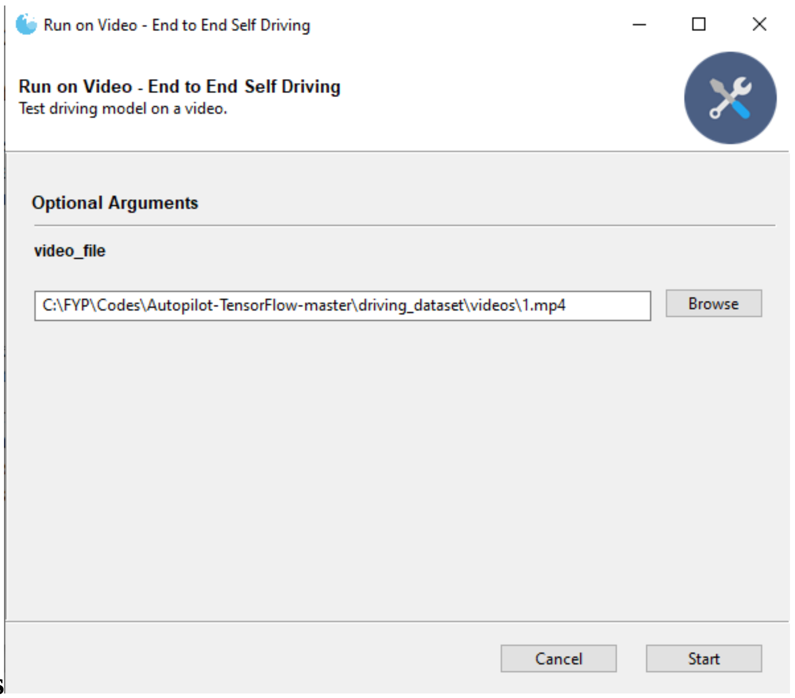
Figure 6: Model Predicting Steering Angle for Given Set of Video Frames

3.6.2.2 Testing on a Video

Run following command in the project directory to test the model on a video:

Python run_video.py0

It will open a window, as shown in figure 7, to select the directory containing the image dataset for testing the model.



Run on Video - End to End Self Driving

Test driving model on a video.

Optional Arguments

video_file

C:\FYP\Codes\Autopilot-TensorFlow-master\driving_dataset\videos\1.mp4

S

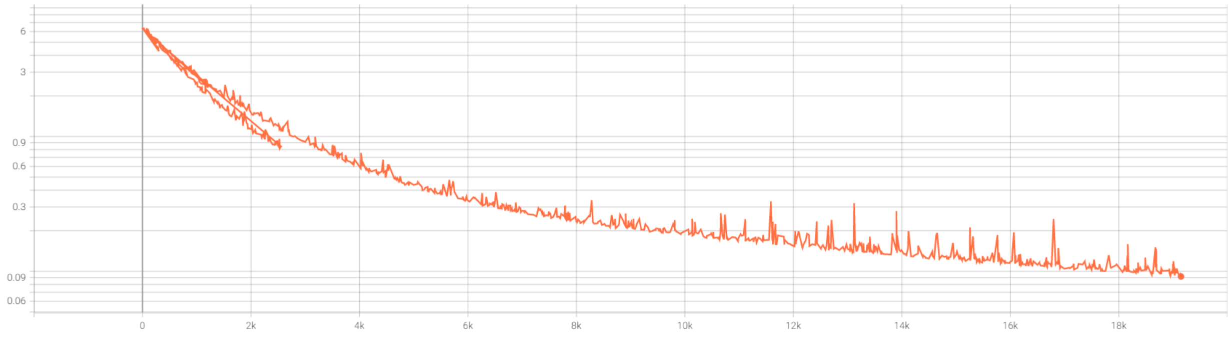
Figure 7: Video Selection to Testing

4 DATA

Dataset videos were recorded by driving around Rancho Palos Verdes and San Pendro California. Dataset size is 3.1 GB and it contains 63,000 images, sampled from the videos with a frame rate of 20 frames/second. It can be downloaded from here.

5 RESULTS

The loss function minimizes the mean square error (MSE) between the steer angle reported by the CNN model and the ground truth steer angle in the data set. The chart below shows that after 30 epochs, the MSE declined from 6.1 to 0.09. The visualization of the steering wheel based on the predicted angle by the model in the training dataset appears fairly smooth and accurate.



6 FUTURE WORK

6.1 REDUCING MSE

6.1.1 Data Augmentation

For increased generalization about unseen data, data augmentation techniques can be applied to simulate different light and weather conditions. Furthermore, GANs can be used to change the environmental conditions without introducing any changes to the symmetry of the road, changing the road structure will make the steering angle mapping inaccurate.

6.1.2 Better Architecture

One obvious next step is to use CNN network to reduce the MSE while keeping an eye on validation set MSE to prevent overfitting. But, an entirely new architecture can be introduced to capture the temporal details of the video as the current CNN model only captures the spatial details from the frames.

One promising direction is to use two-stream CNNs [2] followed by a fully connected layer, further followed by some LSTM layers and finally a fully-connected layer. One CNN model will be fed t^{th} frame and 2^{nd} parallel CNN model will be fed $t+1^{th}$ frame. Using two CNN models in parallel will capture the local-temporal features and the following LSTM layers will capture the global-temporal features.

This approach will especially help with navigating through moving entities on the road, such as cars and humans.

6.2 MULTIPLE MODELS OUT OF THE BOX

More models can be provided within the project to help with establishing comparative benchmarks on a given dataset.

7 CONCLUSION

Without manual breakdown into road or lane marking detection, semantic abstraction, path planning, and control, we have empirically proven that CNNs can learn the entire task of lane and road following. A tiny amount of data from less than a hundred hours of driving was enough to teach the car to operate in a variety of situations, including freeways, local and residential roads, and sunny, cloudy, and wet weather. From a limited training input, the CNN can acquire useful road characteristics. During training, the system learns to recognize the outline of a street without explicit markings.

8 REFERENCES

[1] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... & Zieba, K. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

[2] Weng, X., & Kitani, K. (2019). Learning spatio-temporal features with two-stream deep 3d cnns for lipreading. arXiv preprint arXiv:1905.02540.