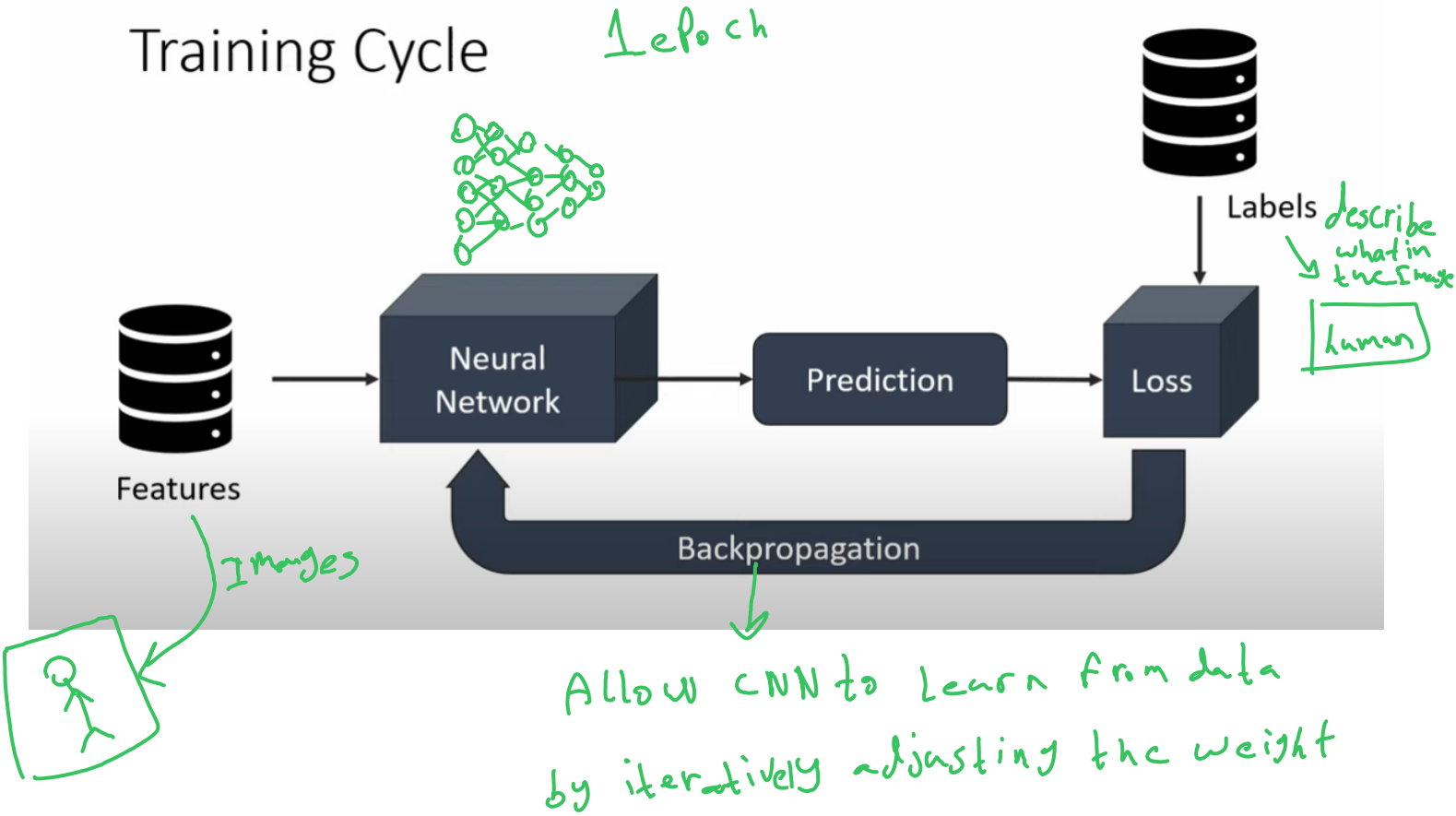

CNN

CNN stands for Convolutional Neural Network. It's a type of deep learning algorithm commonly used for analyzing visual imagery. CNNs are inspired by the organization of the animal visual cortex and have proven to be highly effective in tasks such as image recognition, object detection, and image classification.

Here's a breakdown of CNNs and their components:

Convolutional Layers: These layers apply convolution operations to the input image. Convolution involves sliding a small window called a filter or kernel across the input image and performing element-wise multiplication between the filter and the local region of the image it covers. This operation produces a feature map that highlights patterns or features present in the input image.

Pooling Layers: Pooling layers downsample the feature maps generated by convolutional layers by summarizing local regions. The most common pooling operation is max pooling, which selects the maximum value from each local region. Pooling helps reduce the dimensionality of the feature maps while preserving important features, making the network more computationally efficient and less sensitive to small spatial variations.

Activation Functions: Non-linear activation functions such as ReLU (Rectified Linear Unit) are typically applied after convolutional and pooling layers to introduce non-linearity into the network and enable it to learn complex patterns in the data.

Fully Connected Layers: After several convolutional and pooling layers, the feature maps are flattened into a vector and fed into one or more fully connected layers. These layers perform classification or regression tasks by combining the learned features from previous layers.

Training: CNNs are trained using supervised learning techniques such as backpropagation and gradient descent. During training, the network learns to adjust its parameters (weights and biases) to minimize the difference between predicted and actual outputs.

CNNs have revolutionized many fields, including computer vision, medical image analysis, natural language processing (through techniques like text classification using CNNs), and autonomous vehicles, among others. They are a fundamental tool in the arsenal of deep learning algorithms and have contributed significantly to the advancement of artificial intelligence.

Data augmentation

It is a technique commonly used during the training of machine learning models, especially deep learning models like convolutional neural networks (CNNs). It involves artificially increasing the size of the training dataset by applying various transformations to the existing data samples. These transformations include rotations, translations, flips, changes in scale, brightness adjustments, and more.

There are several reasons why data augmentation is beneficial during training:

Increased Dataset Size:

Improved Generalization:

Regularization:

Reduced Overfitting:

Domain Adaptation:

End to End Learning for Self-Driving Cars

Mariusz Bojarski
NVIDIA Corporation
Holmdel, NJ 07735

Davide Del Testa
NVIDIA Corporation
Holmdel, NJ 07735

Daniel Dworakowski
NVIDIA Corporation
Holmdel, NJ 07735

Bernhard Firner
NVIDIA Corporation
Holmdel, NJ 07735

Beat Flepp
NVIDIA Corporation
Holmdel, NJ 07735

Prasoon Goyal
NVIDIA Corporation
Holmdel, NJ 07735

Lawrence D. Jackel
NVIDIA Corporation
Holmdel, NJ 07735

Mathew Monfort
NVIDIA Corporation
Holmdel, NJ 07735

Urs Muller
NVIDIA Corporation
Holmdel, NJ 07735

Jiakai Zhang
NVIDIA Corporation
Holmdel, NJ 07735

Xin Zhang
NVIDIA Corporation
Holmdel, NJ 07735

Jake Zhao
NVIDIA Corporation
Holmdel, NJ 07735

Karol Zieba
NVIDIA Corporation
Holmdel, NJ 07735

Abstract

We trained a convolutional neural network (CNN) to map raw pixels from a single front-facing camera directly to steering commands. This end-to-end approach proved surprisingly powerful. With minimum training data from humans the system learns to drive in traffic on local roads with or without lane markings and on highways. It also operates in areas with unclear visual guidance such as in parking lots and on unpaved roads.

The system automatically learns internal representations of the necessary processing steps such as detecting useful road features with only the human steering angle as the training signal. We never explicitly trained it to detect, for example, the outline of roads.

Compared to explicit decomposition of the problem, such as lane marking detection, path planning, and control, our end-to-end system optimizes all processing steps simultaneously. We argue that this will eventually lead to better performance and smaller systems. Better performance will result because the internal components self-optimize to maximize overall system performance, instead of optimizing human-selected intermediate criteria, e. g., lane detection. Such criteria understandably are selected for ease of human interpretation which doesn't automatically guarantee maximum system performance. Smaller networks are possible because the system learns to solve the problem with the minimal number of processing steps.

We used an NVIDIA DevBox and Torch 7 for training and an NVIDIA DRIVE™ PX self-driving car computer also running Torch 7 for determining where to drive. The system operates at 30 frames per second (FPS).

1 Introduction

CNNs [1] have revolutionized pattern recognition [2]. Prior to the widespread adoption of CNNs, most pattern recognition tasks were performed using an initial stage of hand-crafted feature extraction followed by a classifier. The breakthrough of CNNs is that features are learned automatically from training examples. The CNN approach is especially powerful in 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.

While CNNs with learned features have been in commercial use for over twenty years [3], their adoption has exploded in the last few years because of two recent developments. First, large, labeled data sets such as the Large Scale Visual Recognition Challenge (ILSVRC) [4] have become available for training and validation. Second, CNN learning algorithms have been implemented on the massively parallel graphics processing units (GPUs) which tremendously accelerate learning and inference.

In this paper, we describe a CNN that goes beyond pattern recognition. It learns the entire processing pipeline needed to steer an automobile. The groundwork for this project was done over 10 years ago in a Defense Advanced Research Projects Agency (DARPA) seedling project known as DARPA Autonomous Vehicle (DAVE) [5] in which a sub-scale radio control (RC) car drove through a junk-filled alley way. DAVE was trained on hours of human driving in similar, but not identical environments. The training data included video from two cameras coupled with left and right steering commands from a human operator.

In many ways, DAVE-2 was inspired by the pioneering work of Pomerleau [6] who in 1989 built the Autonomous Land Vehicle in a Neural Network (ALVINN) system. It demonstrated that an end-to-end trained neural network can indeed steer a car on public roads. Our work differs in that 25 years of advances let us apply far more data and computational power to the task. In addition, our experience with CNNs lets us make use of this powerful technology. (ALVINN used a fully-connected network which is tiny by today's standard.)

While DAVE demonstrated the potential of end-to-end learning, and indeed was used to justify starting the DARPA Learning Applied to Ground Robots (LAGR) program [7], DAVE's performance was not sufficiently reliable to provide a full alternative to more modular approaches to off-road driving. DAVE's mean distance between crashes was about 20 meters in complex environments.

Nine months ago, a new effort was started at NVIDIA that sought to build on DAVE and create a robust system for driving on public roads. The primary motivation for this work is to avoid the need to recognize specific human-designated features, such as lane markings, guard rails, or other cars, and to avoid having to create a collection of "if, then, else" rules, based on observation of these features. This paper describes preliminary results of this new effort.

2 Overview of the DAVE-2 System

Figure 1 shows a simplified block diagram of the collection system for training data for DAVE-2. Three cameras are mounted behind the windshield of the data-acquisition car. Time-stamped video from the cameras is captured simultaneously with the steering angle applied by the human driver. This steering command is obtained by tapping into the vehicle's Controller Area Network (CAN) bus. In order to make our system independent of the car geometry, we represent the steering command as $1/r$ where r is the turning radius in meters. We use $1/r$ instead of r to prevent a singularity when driving straight (the turning radius for driving straight is infinity). $1/r$ smoothly transitions through zero from left turns (negative values) to right turns (positive values).

Training data contains single images sampled from the video, paired with the corresponding steering command ($1/r$). Training with data from only the human driver is not sufficient. The network must learn how to recover from mistakes. Otherwise the car will slowly drift off the road. The training data is therefore augmented with additional images that show the car in different shifts from the center of the lane and rotations from the direction of the road.

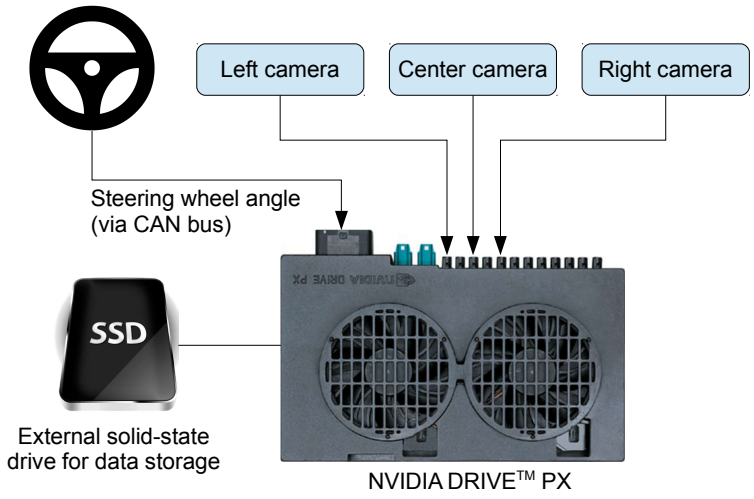

Figure 1: High-level view of the data collection system.

Images for two specific off-center shifts can be obtained from the left and the right camera. Additional shifts between the cameras and all rotations are simulated by viewpoint transformation of the image from the nearest camera. Precise viewpoint transformation requires 3D scene knowledge which we don't have. We therefore approximate the transformation by assuming all points below the horizon are on flat ground and all points above the horizon are infinitely far away. This works fine for flat terrain but it introduces distortions for objects that stick above the ground, such as cars, poles, trees, and buildings. Fortunately these distortions don't pose a big problem for network training. The steering label for transformed images is adjusted to one that would steer the vehicle back to the desired location and orientation in two seconds.

A block diagram of our training system is shown in Figure 2. 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 as implemented in the Torch 7 machine learning package.

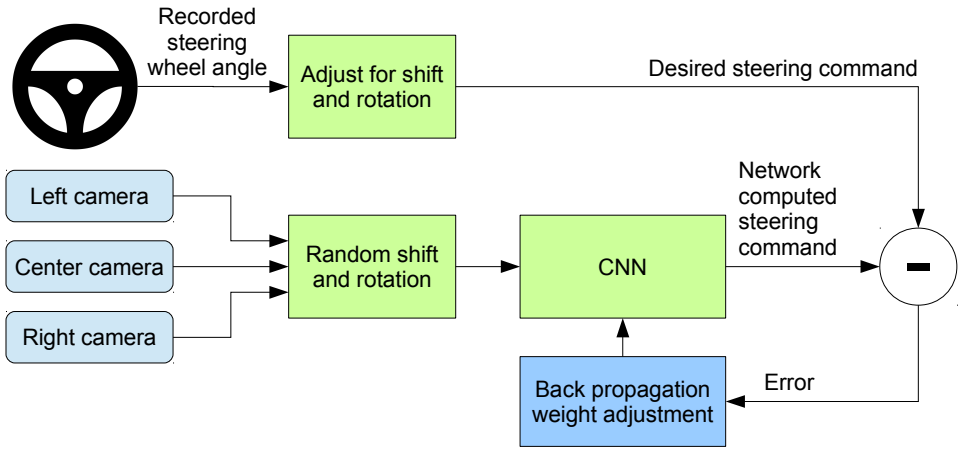

Figure 2: Training the neural network.

Once trained, the network can generate steering from the video images of a single center camera. This configuration is shown in Figure 3.

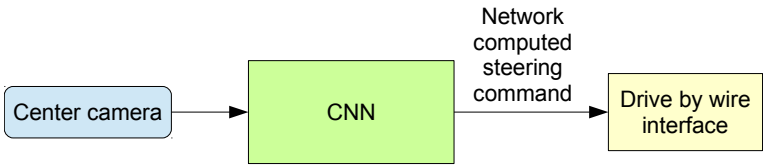

Figure 3: The trained network is used to generate steering commands from a single front-facing center camera.

3 Data Collection

Training data was collected by driving on a wide variety of roads and in a diverse set of lighting and weather conditions. Most road data was collected in central New Jersey, although highway data was also collected from Illinois, Michigan, Pennsylvania, and New York. Other road types include two-lane roads (with and without lane markings), residential roads with parked cars, tunnels, and unpaved roads. Data was collected in clear, cloudy, foggy, snowy, and rainy weather, both day and night. In some instances, the sun was low in the sky, resulting in glare reflecting from the road surface and scattering from the windshield.

Data was acquired using either our drive-by-wire test vehicle, which is a 2016 Lincoln MKZ, or using a 2013 Ford Focus with cameras placed in similar positions to those in the Lincoln. The system has no dependencies on any particular vehicle make or model. Drivers were encouraged to maintain full attentiveness, but otherwise drive as they usually do. As of March 28, 2016, about 72 hours of driving data was collected.

4 Network Architecture

We train 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 (see Section 5.2). Our network architecture is shown in Figure 4. The network consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. The input image is split into YUV planes and passed to the network.

The first layer of the network performs image normalization. The normalizer is hard-coded and is not adjusted in the learning process. Performing normalization in the network allows the normalization scheme to be altered with the network architecture and to be accelerated via GPU processing.

The convolutional layers were designed to perform feature extraction and were chosen empirically through a series of experiments that varied layer configurations. We use strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel and a non-strided convolution with a 3×3 kernel size in the last two convolutional layers.

We follow the five convolutional layers with three fully connected layers leading to an output control value which is the inverse turning radius. The fully connected layers are designed to function as a controller for steering, but we note that by training the system end-to-end, it is not possible to make a clean break between which parts of the network function primarily as feature extractor and which serve as controller.

5 Training Details

5.1 Data Selection

The first step to training a neural network is selecting the frames to use. Our collected data is labeled with road type, weather condition, and the driver's activity (staying in a lane, switching lanes, turning, and so forth). To train a CNN to do lane following we only select data where the driver was staying in a lane and discard the rest. We then sample that video at 10 FPS. A higher sampling rate would result in including images that are highly similar and thus not provide much useful information.

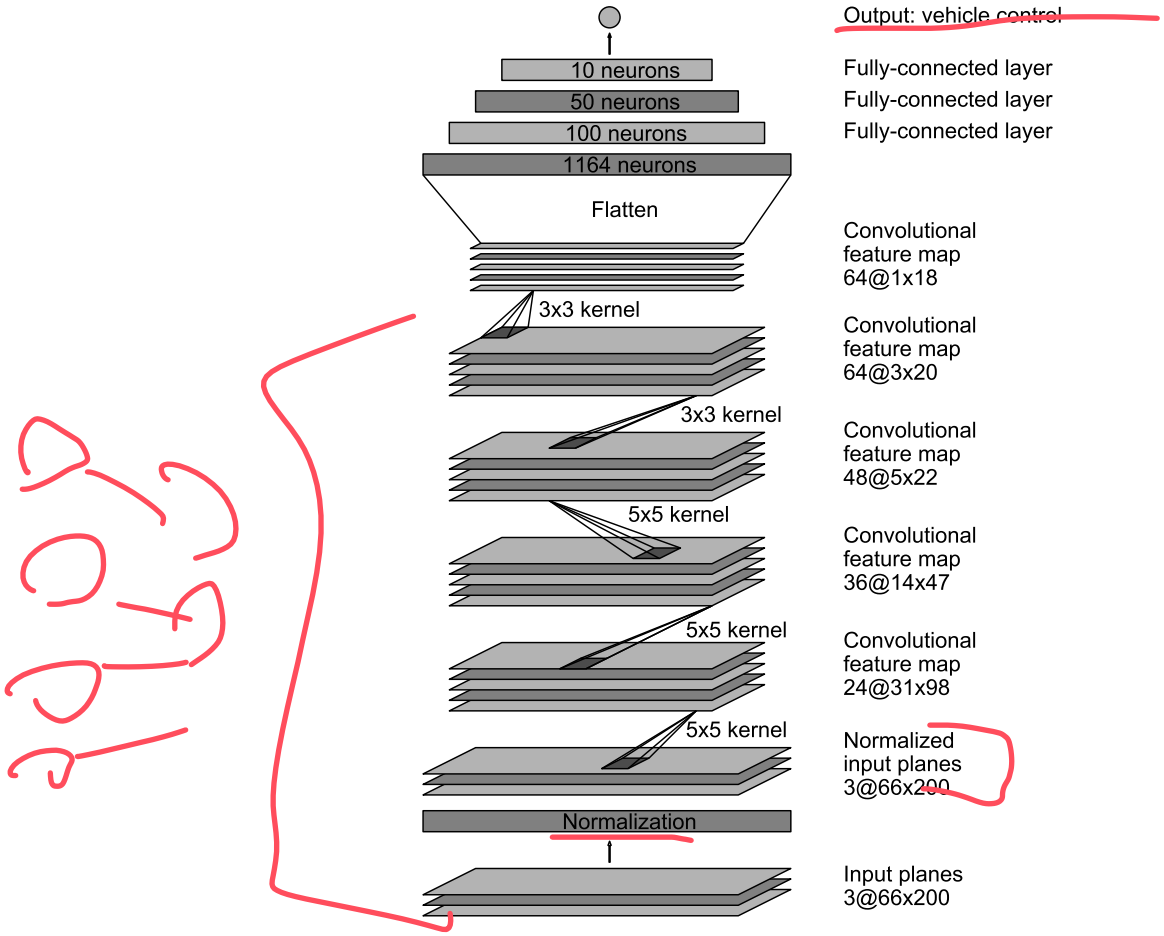

Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

To remove a bias towards driving straight the training data includes a higher proportion of frames that represent road curves.

5.2 Augmentation

After selecting the final set of frames we augment the data by adding artificial shifts and rotations to teach the network how to recover from a poor position or orientation. The magnitude of these perturbations is chosen randomly from a normal distribution. The distribution has zero mean, and the standard deviation is twice the standard deviation that we measured with human drivers. Artificially augmenting the data does add undesirable artifacts as the magnitude increases (see Section 2).

6 Simulation

Before road-testing a trained CNN, we first evaluate the networks performance in simulation. A simplified block diagram of the simulation system is shown in Figure 5.

The simulator takes pre-recorded videos from a forward-facing on-board camera on a human-driven data-collection vehicle and generates images that approximate what would appear if the CNN were, instead, steering the vehicle. These test videos are time-synchronized with recorded steering commands generated by the human driver.

Since human drivers might not be driving in the center of the lane all the time, we manually calibrate the lane center associated with each frame in the video used by the simulator. We call this position the "ground truth".

The simulator transforms the original images to account for departures from the ground truth. Note that this transformation also includes any discrepancy between the human driven path and the ground truth. The transformation is accomplished by the same methods described in Section 2.

The simulator accesses the recorded test video along with the synchronized steering commands that occurred when the video was captured. The simulator sends the first frame of the chosen test video, adjusted for any departures from the ground truth, to the input of the trained CNN. The CNN then returns a steering command for that frame. The CNN steering commands as well as the recorded human-driver commands are fed into the dynamic model [8] of the vehicle to update the position and orientation of the simulated vehicle.

The simulator then modifies the next frame in the test video so that the image appears as if the vehicle were at the position that resulted by following steering commands from the CNN. This new image is then fed to the CNN and the process repeats.

The simulator records the off-center distance (distance from the car to the lane center), the yaw, and the distance traveled by the virtual car. When the off-center distance exceeds one meter, a virtual human intervention is triggered, and the virtual vehicle position and orientation is reset to match the ground truth of the corresponding frame of the original test video.

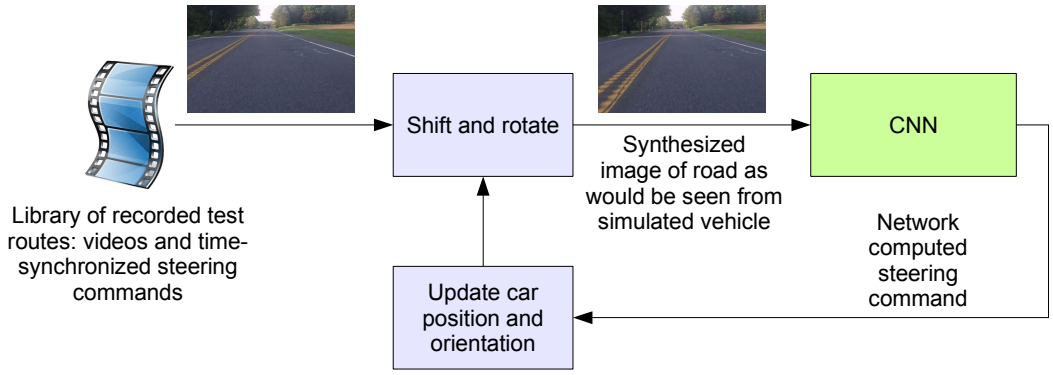

Figure 5: Block-diagram of the drive simulator.

7 Evaluation

Evaluating our networks is done in two steps, first in simulation, and then in on-road tests.

In simulation we have the networks provide steering commands in our simulator to an ensemble of prerecorded test routes that correspond to about a total of three hours and 100 miles of driving in Monmouth County, NJ. The test data was taken in diverse lighting and weather conditions and includes highways, local roads, and residential streets.

7.1 Simulation Tests

We estimate what percentage of the time the network could drive the car (autonomy). The metric is determined by counting simulated human interventions (see Section 6). These interventions occur when the simulated vehicle departs from the center line by more than one meter. We assume that in real life an actual intervention would require a total of six seconds: this is the time required for a human to retake control of the vehicle, re-center it, and then restart the self-steering mode. We calculate the percentage autonomy by counting the number of interventions, multiplying by 6 seconds, dividing by the elapsed time of the simulated test, and then subtracting the result from 1:

$$\text{autonomy} = (1 - \frac{(\text{number of interventions}) \cdot 6 \text{ seconds}}{\text{elapsed time [seconds]}}) \cdot 100 \quad (1)$$

Figure 6: Screen shot of the simulator in interactive mode. See Section 7.1 for explanation of the performance metrics. The green area on the left is unknown because of the viewpoint transformation. The highlighted wide rectangle below the horizon is the area which is sent to the CNN.

Thus, if we had 10 interventions in 600 seconds, we would have an autonomy value of

$$(1 - \frac{10 \cdot 6}{600}) \cdot 100 = 90\%$$

7.2 On-road Tests

After a trained network has demonstrated good performance in the simulator, the network is loaded on the DRIVE^{TM} PX in our test car and taken out for a road test. For these tests we measure performance as the fraction of time during which the car performs autonomous steering. This time excludes lane changes and turns from one road to another. For a typical drive in Monmouth County NJ from our office in Holmdel to Atlantic Highlands, we are autonomous approximately 98% of the time. We also drove 10 miles on the Garden State Parkway (a multi-lane divided highway with on and off ramps) with zero intercepts.

A video of our test car driving in diverse conditions can be seen in [9].

7.3 Visualization of Internal CNN State

Figures 7 and 8 show the activations of the first two feature map layers for two different example inputs, an unpaved road and a forest. In case of the unpaved road, the feature map activations clearly show the outline of the road while in case of the forest the feature maps contain mostly noise, i. e., the CNN finds no useful information in this image.

This demonstrates that the CNN learned to detect useful road features on its own, i. e., with only the human steering angle as training signal. We never explicitly trained it to detect the outlines of roads, for example.

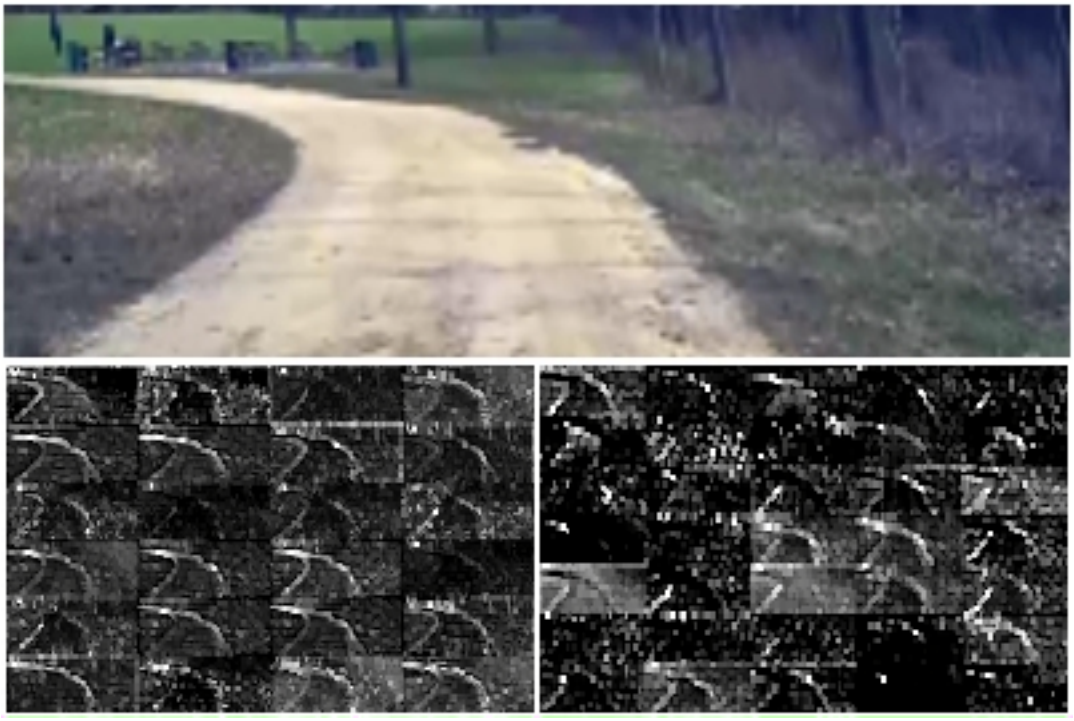

Figure 7: How the CNN "sees" an unpaved road. Top: subset of the camera image sent to the CNN. Bottom left: Activation of the first layer feature maps. Bottom right: Activation of the second layer feature maps. This demonstrates that the CNN learned to detect useful road features on its own, i. e., with only the human steering angle as training signal. We never explicitly trained it to detect the outlines of roads.

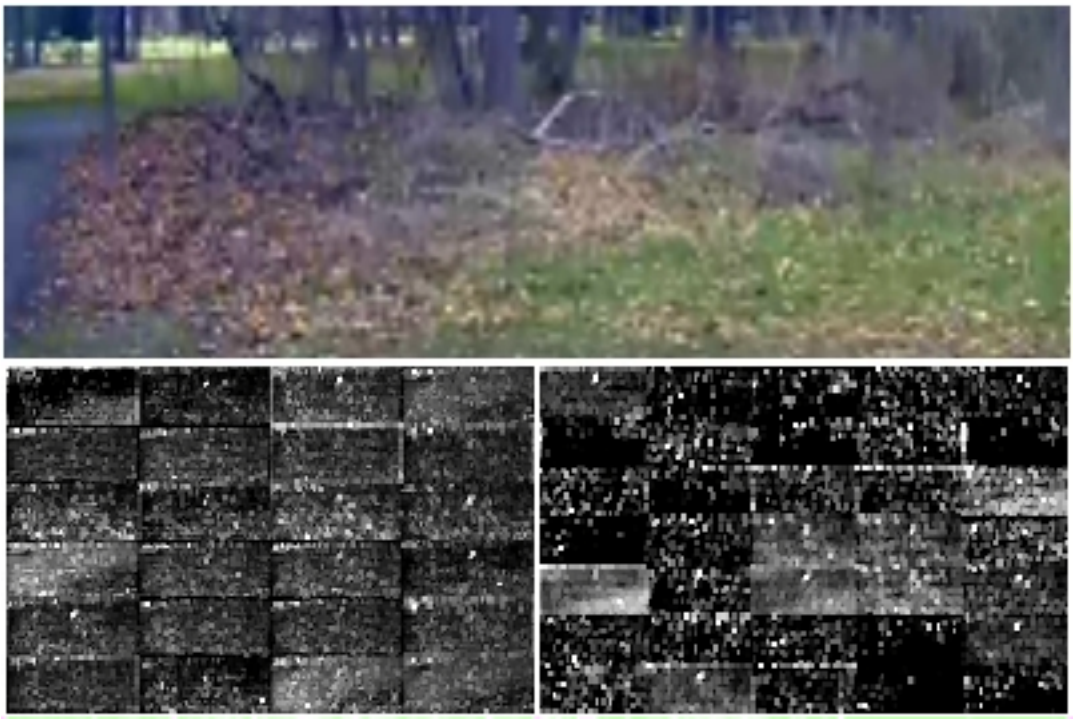

Figure 8: Example image with no road. The activations of the first two feature maps appear to contain mostly noise, i. e., the CNN doesn't recognize any useful features in this image.

8 Conclusions

We have empirically demonstrated that CNNs are able to learn the entire task of lane and road following without manual decomposition into road or lane marking detection, semantic abstraction, path planning, and control. A small amount of training data from less than a hundred hours of driving was sufficient to train the car to operate in diverse conditions, on highways, local and residential roads in sunny, cloudy, and rainy conditions. The CNN is able to learn meaningful road features from a very sparse training signal (steering alone).

The system learns for example to detect the outline of a road without the need of explicit labels during training.

More work is needed to improve the robustness of the network, to find methods to verify the robustness, and to improve visualization of the network-internal processing steps.

References

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Winter 1989. URL: http://yann.lecun.org/exdb/publis/pdf/lecun-89e.pdf.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[3] L. D. Jackel, D. Sharman, Stenard C. E., Strom B. I., , and D Zuckert. Optical character recognition for self-service banking. *AT&T Technical Journal*, 74(1):16–24, 1995.

[4] Large scale visual recognition challenge (ILSVRC). URL: http://www.image-net.org/challenges/LSVRC/.

[5] Net-Scale Technologies, Inc. Autonomous off-road vehicle control using end-to-end learning, July 2004. Final technical report. URL: http://net-scale.com/doc/net-scale-dave-report.pdf.

[6] Dean A. Pomerleau. ALVINN, an autonomous land vehicle in a neural network. Technical report, Carnegie Mellon University, 1989. URL: http://repository.cmu.edu/cgi/viewcontent.cgi?article=2874&context=compsci.

[7] Wikipedia.org. DARPA LAGR program. http://en.wikipedia.org/wiki/DARPA_LAGR_Program.

[8] Danwei Wang and Feng Qi. Trajectory planning for a four-wheel-steering vehicle. In *Proceedings of the 2001 IEEE International Conference on Robotics & Automation*, May 21–26 2001. URL: http://www.ntu.edu.sg/home/edwwang/confpapers/wdwicar01.pdf.

[9] DAVE 2 driving a lincoln. URL: https://drive.google.com/open?id=0B9raQzOpizn1TkRIa241ZnBEcjQ.