# Teaching Cars to Steer Themselves With Deep Learning


## Ian Timmis, Nicholas Paul, CJ Chung

**College of Arts and Sciences, Lawrence Technological University**


## INTRODUCTION

Traditional approaches for steering a vehicle using machine vision require large amounts of robust hand-crafted software which is both time consuming and expensive. The presented method uses a deep neural network to teach cars to steer themselves without any additional software. We created a labeled dataset for the ACTor (Autonomous Campus TranspORt) electric vehicle [1] by pairing real world images taken during a drive with the associated steering wheel angle. We trained a model end to end using modern deep learning techniques including convolutional neural networks and transfer learning [2] to automatically detect relevant features in the input and provide a predicted output. This means that no traditional hand engineered algorithm features were required for this implementation. We currently use an pretrained inception network on the ImageNet dataset to leverage the high level features learned from ImageNet to the steering problem through transfer learning. We removed the top portion of the network and replaced it with a linear regression node to provide the output. The model is trained end to end using backpropagation. The trained model is integrated with vehicle software on ROS (Robot Operating System) [3] to read image data and provide a corresponding steering angle in real time. The current model achieves 15.2 degree error on average. As development continues the model may replace the current lane centering software and will be used for IGVC Self-Drive competition and campus transportation. Fig. 1 shows the training process.

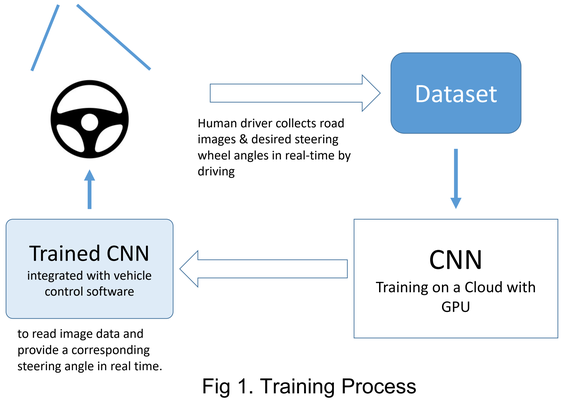



Fig 1. Training Process

## DATA ACQUISITION

The model uses an end-to-end architecture which means that the neural network is responsible for converting an input image from the dashboard camera directly into a steering wheel command. In order to collect training data, a ROS node was added to the existing ACTor Autonomous System that recorded a frame from the dashboard camera as well as a steering wheel angle from the drive-by-wire vehicle reporting subsystem. (See Fig. 3). The node was simple to add due to the modular framework of the existing system [1].

Fig. 2 shows a few examples of images and corresponding steering wheel angles. The data collection node captured new images and angles about 15 times per second over several routes on campus for a total of a few thousand data points.

ACTor was driven around several roads on campus to collect a large range of steering angles. Locations with many turns were selected to reduce bias toward driving straight. Additionally, routes were driven in both directions to reduce bias for steering in only one direction.

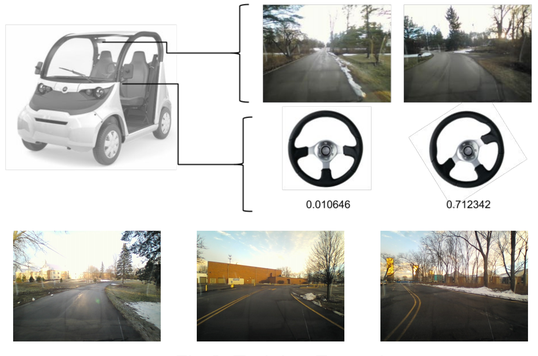



Fig 2. Training Examples

## DEVELOPMENT

We developed this model using the Python programming language with the TensorFlow and Keras deep learning libraries. The model was trained on the FloydHub cloud computing service to utilize the speedup acquired by using GPU for deep learning. GPU has become the primary way of training deep learning models because they can massively parallelize the computations over all it's cores. Deep neural networks are able to be parallelized because of they are largely comprised of matrix operations. Operations can be computed on multiple elements of a matrix simultaneously. Training took approximately 3 hours to complete.

We use a 60/20/20 split on our 4600 image dataset into a training set, validation set, and test set. By using both a validation and a test set to evaluate our model, we can greatly increase our confidence that the model will generalize well and not overfit the test set. The labels are converted from radians into degrees before being used to facilitate the use of our loss function. All of the pixel values are scaled to values between 0 and 1 using MinMax normalization.

We utilize a Convolutional Neural Network (Inceptionv3) pretrained on the ImageNet dataset as the base of our model. We remove the logistic layers specific to the ImageNet vision problem and replace them with our own 1024-neuron fully connected layer followed by a linear regression node to predict the steering wheel angle. When training the model, we transfer knowledge from the steering problem to this model through stages of Transfer Learning and Fine-Tuning [2]. We train the model stochastically using the gradient based optimization algorithm, Adam, to minimize the Mean Squared Error of predicted outputs. We utilize Early Stopping to stop training when learning has stagnated. To ensure that the minimum loss is selected for, we also utilize Model Checkpointing to make sure the best weights are saved.

## INTEGRATION

The deep learning subsystem is integrated in two ways: data acquisition and model evaluation. These functions interact with the ACTor system independently of one another. First, the data collection node is used to collect data for training (See section "Data Acquisition"). The training data is used by custom training software independent of the vehicle entirely. This software outputs a model file which contains the parameters for the deep neural network. Finally, the model is loaded onto the vehicle by the steering node (Fig. 3) which captures an image from the dashboard camera, evaluates the model on the image, and then outputs a steering angle in real time. The steering angle is sent to the steering wheel driver of the drive by wire subsystem.

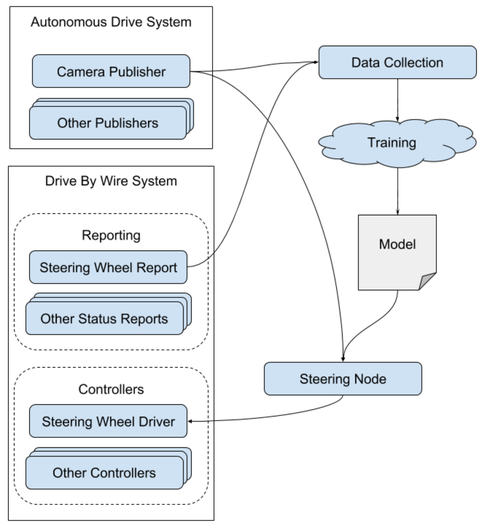
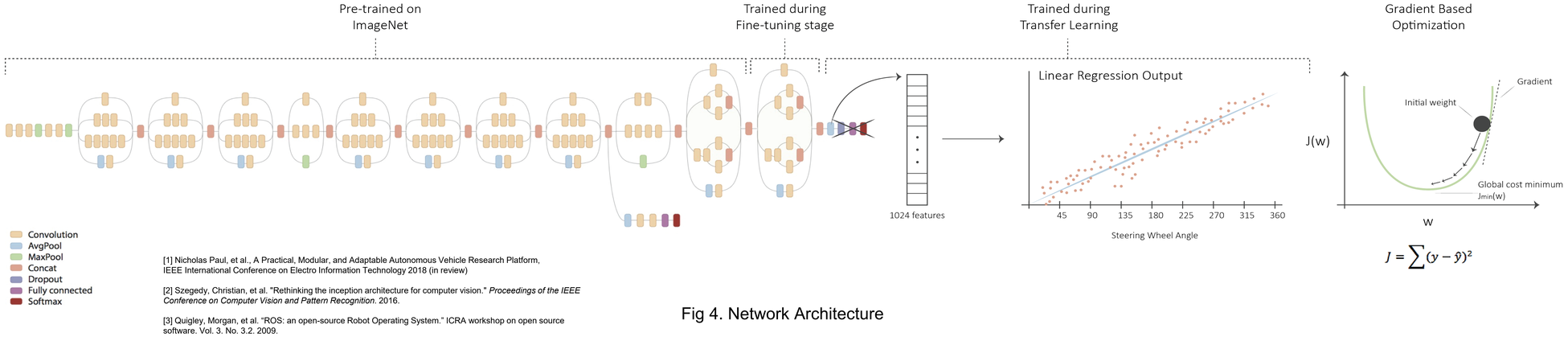



Fig 3. ROS Integration

## ALGORITHM





[1] Nicholas Paul, et al., A Practical, Modular, and Adaptable Autonomous Vehicle Research Platform, IEEE International Conference on Electro Information Technology 2018 (in review)

[2] Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[3] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.


Fig 4. Network Architecture

## RESULTS

After training the model, we find that it achieves a test loss of ~233 which equates to the predicted steering angle yielding ~15.2 degrees of error on average. We speculate that gathering more data will help greatly reduce the error of our model.

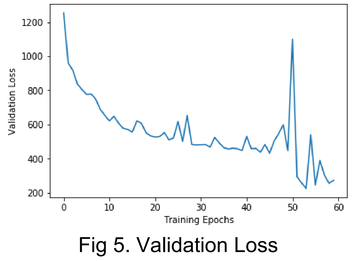



Fig 5. Validation Loss