Steering angle prediction for Self-driving cars from 2D Images

Prakash Baskaran[1] and Dr. Carlos Morato[2]

***Abstract*— Self-driving cars are already a reality and they are no longer a science fiction. They are soon expected to disrupt the automobile industry that we know of today. Udacity has an on-going self-driving car challenge aimed to predict steering angles based on only the images we get from a single monocular camera placed at the center of the windshield. Udacity has released a dataset of images captured while driving labeled with the corresponding steering angle.**

In this paper, we propose two deep learning models to learn and predict the steering angle based on the input images. The first approach is to come up with an ingenious deep neural network model that includes Convolutional Neural Networks, Dropout [1] and Network concatenation. The second approach is to apply Transfer Learning method and train the last few layers on an already existing model.

I. INTRODUCTION

The growth of self-driving cars has expanded drastically over the last decade. They are soon expected to be seen on roads and are going to revolutionize the automotive market that we know of today. It is turning out to be the current topic of research interest across the globe and it is no wonder that all tech giants like Google, NVIDIA, Intel, Tesla, Mercedes, BMW etc. are making enormous efforts to encapsulate this technology. Udacity has an ongoing challenge to create an open-source platform for a self-driving car. The second challenge for the Udacity initiative is to teach a car how to drive using the images from a monocular camera placed in the front windshield. Udacity has released a dataset [2] of images taken while driving along with the corresponding steering angle data. The objective of this project is to build a deep learning system that is capable of driving in many different situations such as varying weather conditions, avoid an obstacle and even going off-road. The MSE (Mean Squared Error) minimization standard is used to benchmark the results with the predicted output against the ground truth. In this project, we explore a variety of deep learning techniques such as 2D convolutional neural networks, Dropouts and transfer learning, etc. to predict the steering angle.

We aim at providing a robust end-to-end solution to this problem similar to what NVIDIA has done recently [3]. The catch here is to eliminate the need for hand-crafted features and instead create a system that first learns the features necessary to predict the steering angle and then trains itself overtime to drive by observing these features. The motivation behind using Convolutional Neural Networks is that they learn features automatically from the training examples and are extremely powerful in image recognition and classification [4].

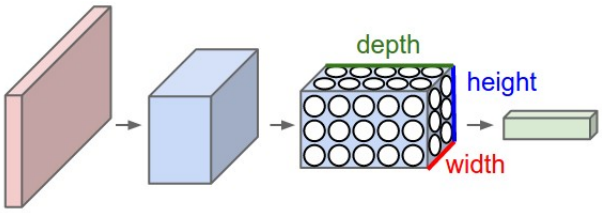


Fig. 1: Model of a CNN

The two approaches that will be exploring are first, a model that makes use of Convolutional Neural Networks and Network concatenation. We propose two different networks, a wide network and a narrow network, of which one the wide network is 4 layers deeper than the other. Both these networks are trained on the same input data and run in parallel to one other. The key idea here is to merge two different CNNs together to produce a hybrid network so that the result is a network that learns all the high level features but also accounts for the low level features learned by the narrow network. The second approach is to use lower layers of a high-quality model already trained on a similar dataset, in this case ImageNet [5]. This method is commonly referred as Transfer Learning. This method proves to be extremely useful when the amount of data available to train the model is low.

A. Literature Review

- The use of neural network in the field of autonomous navigation was spearheaded by D.A Pomerleau who built the Autonomous Land Vehicle in a Neural Network (ALVINN) [6] system in the year 1989. The model takes input from a camera and LIDAR and produces the direction the vehicle should travel in order to follow the road. The ALVINN is relatively a simple 3-layer neural network (Fig. 2) designed for the task of road following. Nevertheless, this model demonstrated the potential of neural networks in the field of autonomous navigation.
- Recently, NVIDIA published a paper [3] on end-to-end deep learning for self-driving cars. The paper discusses a basic CNN model (Fig. 3) to map raw pixels from a single front-facing camera directly to steering

[1]Prakash Baskaran is a Graduate Student of the Robotics Engineering Department, Worcester Polytechnic Institute, Worcester, MA 01609, USA. e-mail: pbaskaran@wpi.edu

[2]C. W. Morato is with the Robotics Department, Worcester Polytechnic Institute, Worcester, MA 01609 USA. He is also with the Department of Mechatronics and Sensors, US Corporate Research Center, ABB Inc., CT 06002 USA e-mail: cwmorato@wpi.edu, carlos.morato@us.abb.com

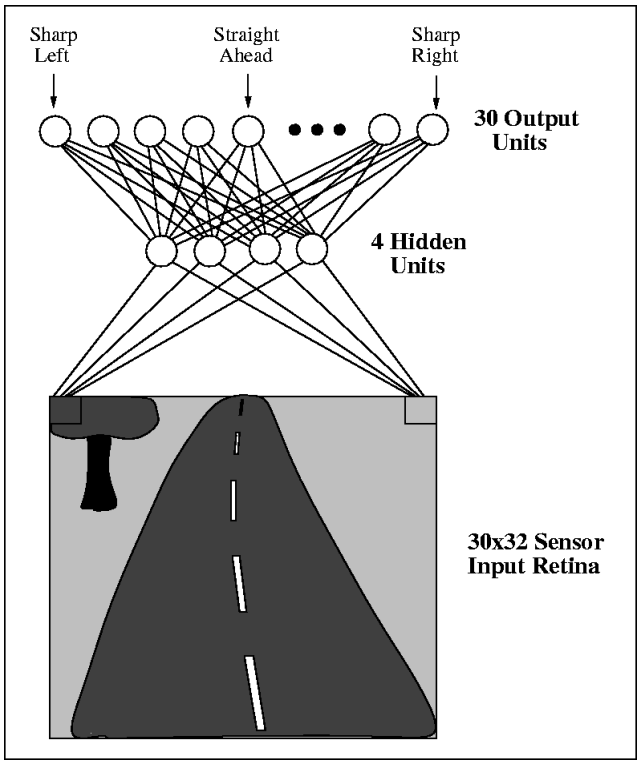


Fig. 2: ALVINN: Simple 3-layered Neural Network

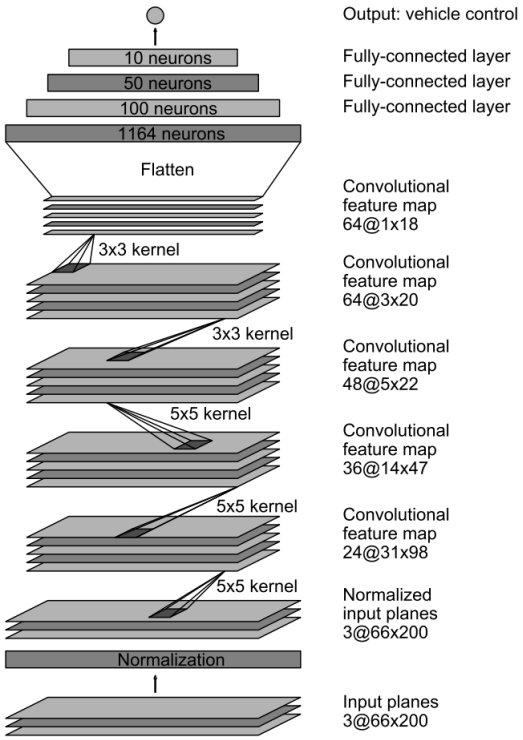


Fig. 3: CNN Architecture used by NVIDIA

commands. The system was never explicitly trained to detect the outline of roads. But interestingly the system was capable of automatically learning internal representations of the necessary processing steps such as detecting useful road features. Compared to explicit decomposition of the problem, such as lane marking detection, path planning, and control, the end-to-end system optimizes all processing steps simultaneously. This laid a strong foundation for end-to-end deep learning models for autonomous navigation.

- Although deep convolutional neural networks have greatly advanced the performance of visual-recognition systems, they take a lot of time to be trained. And deeper the neural network gets, the more difficult it is to train them. Microsoft Research Asia had come up with a framework called Deep Residual Learning [7] that mitigates the problem of training deep neural networks. The idea of residual mapping connection is to use network layers to fit a residual mapping instead of directly trying to fit desired underlying mapping. Residual networks ease the training of networks and at the same time, they are substantially deeper than those networks used previously.

B. Problem Description

To emulate human driving patterns in an autonomous vehicle that is capable of driving in many different situations in real time, taking input only from a single monocular camera, using deep learning techniques.

II. Overview

We have developed two types of model. The first one uses 2D ConvNets followed by network concatenation. The second model uses transfer learning on a pre-trained model where the lower layers are blocked from training.

A. 2D Convolutional Model

Few main types of layers that we have made use are Convolutional layer, Dropout layer, Merge layer and Fully Connected layer.

- The convolutional layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.
- The dropout layer drops out a random set of activations in that layer by setting them to zero in the forward pass. This forces the network to be able to provide the right output even if some of the activations are dropped out. This ensures that the network is not overfitting the data.
- The Merge layer is used to concatenate outputs coming from different layers during the forward pass and splits the gradient during the backward pass.
- The fully connected layers will output the final prediction. It works like an ordinary neural network where each neuron in the current layer is connected to all the neurons in the previous layer.

B. Transfer Learning

Transfer learning is a way of using high quality models that were trained on existing large datasets. The idea of transfer learning is that features learned in the lower layers of the model are likely transferable to another dataset. These lower level features would be useful in the new dataset such as edges.

We make use of two models that performed exceptionally well in the imagenet dataset. First one is the InceptionV3 [8] model by Google DeepMind and second one is the ResNet50 [7] by Microsoft Research Asia.

- The hypothesis behind inception blocks is to have networks that are deeper but also at the same time they are designed to perform well even under strict memory constraints and computational budget.
- The key idea behind residual connection is to fit a residual mapping instead of directly trying to fit the desired mapping.
  Without the residual connection:

$$F(x) = H(G(x)) \tag{1}$$

  With the residual connection:

$$F(x) = H(G(x)) - G(x) - x \tag{2}$$

  This skipping of gradient over different layers alleviates problems with the back propagation algorithm in deep networks such as the vanishing gradient problem.

III. Dataset

The dataset we used is provided by Udacity. Time-stamped video from the camera 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 the system independent of the car geometry, they represent the steering command as $1/r$, where r is the turning radius in meters. They 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 negative values (left turns) to positive values (right turns). The training images come from 5 different driving scenarios:

- 221 seconds, direct sunlight, many lighting changes. Good turns in beginning, discontinuous shoulder lines, ends in lane merge, divided highway.
- Discontinuous shoulder lines, ends in lane merge, divided highway 791 seconds, two lane road, shadows are prevalent, traffic signal (green), very tight turns where center camera cant see much of the road, direct sunlight, fast elevation changes leading to steep gains/losses over summit. Turns into divided highway around 350s, quickly returns to 2 lanes.
- 99 seconds, divided highway segment of return trip over the summit.
- 212 seconds, guardrail and two lane road, shadows in beginning may make training difficult, mostly normalizes towards the end.

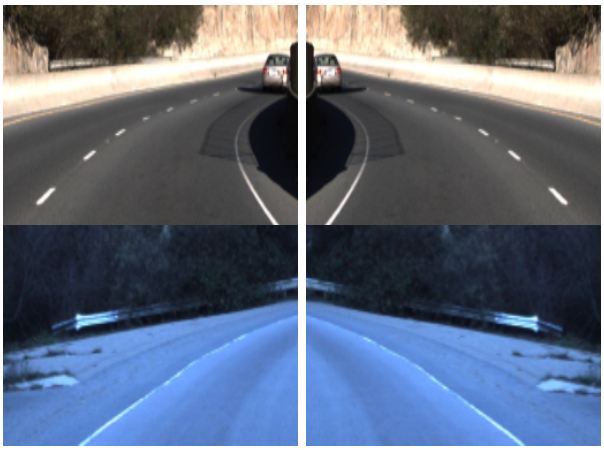


Fig. 4: Flipped images

- 371 seconds, divided multi-lane highway with a fair amount of traffic.

The frame rate of these videos are 20fps and the resolution of each frame is 640x480. Training data consists of 67574 frames obtained from the videos mentioned above and the corresponding labels including steering angle, torque applied and the vehicle velocity. We further split this data into training, validation and test data in a 60/20/20 fashion.

A. Data Augmentation and Preprocessing

We flip the image vertically (Fig. 4) by taking its transpose and label the steering angle of the flipped image to be the negative of steering angle of the original image. This helps us in increasing the total number of images to two times the original count. We also randomly take some images and change the brightness of the image by altering the HSV scale to increase the density of the dataset.

For each image, we normalize the pixel values from [0,255] to [0,1] by dividing the image matrix by 255. We further resize the image to 192x256 (rescaling by a factor of 2.5) for the 2D Convolutional model and to 224x224 for the transfer learning model.

IV. Network Architecture

We use 2 approaches and observe how well does each of these scales up for the given task. In both the models, the weights are being trained to minimize the mean squared error between the steering angle of the human driver and the steering angle predicted by the model. This loss function is common for regression problems. This function is simply the mean of the sum of the squared difference between the actual and predicted result, taken in batches. We use Adaptive Moment Estimation (Adam) Optimizer since it works well in practice when compared to other optimizing algorithms.

A. 2D Convolutional Model

In this approach we make use of 2 different models both of which are trained on the same input data and they both run in parallel to each other. They are finally merged together to produce the steering angle as output. For comparison we

present the results of all the 3 models (narrow, wide and merged model) individually and see how the merged model fares well.

In order to establish a baseline for our research, we use the architecture from Comma AI [9] for the narrow model and the architecture from NVIDIA [3] for the wide model. The narrow model comprises of 3 convolutional layers and 2 fully connected layers as shown in Fig. 5. The first layer uses 16 filters each of kernel size 8x8 with a stride of 4. The second layer uses 32 filters with a kernel size of 5x5 and stride of 2. The last convolutional layer has 64 filters with a kernel size of 5x5 and stride 2 convolution. The output of the last convolutional layer is flattened and connected to 2 fully connected layers with units 512 and 1, respectively with a dropout layer immediately before and after the 512 units fully connected layer. Dropout prevents the network from overfitting the data by not letting the units adapt too much. All these layers use ReLU as their activation.

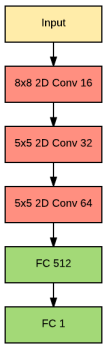


Fig. 5: Narrow CNN Architecture

The wide model comprises of 5 convolutional layers and 4 fully connected layers as shown in Fig. 6. All the convolutions are strided with a size of 2. The first layer uses 24 filters with a of kernel size 5x5. The second layer uses 36 filters with a kernel size of 5x5. The third layer uses 48 filters with a kernel size of 5x5. The last two convolutional layers have 64 filters each with a kernel size of 3x3. The output of the last convolutional layer is flattened and connected to 4 fully connected layers with units 100, 50, 10 and 1, respectively. Dropout layers are added immediately after each convolutional layer. All these layers use ReLU as their activation.

The merged layer takes the input data and multiplexes them into two separate channels and then merges again into one. The basic structure of the merged model comes from the narrow model and the wide model. The convolutional outputs of both the models are flattened concatenated together through a merge layer. The merged layer is then followed by 4 fully connected layers as shown in Fig. 7. The motivation behind using the merged model is to not lose out on the basic essential features as we go deeper and deeper. Hence, we come up two models, one wider and the other narrower,so that the wide model learns all the high level features and the

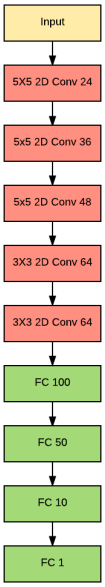


Fig. 6: Wide CNN Architecture

narrow model learns the low level features. This increases the robustness of the system and the output is able to generalize well for all different kinds of input.

B. Transfer Learning Model

For this model, we used the idea of transfer learning. Of the pre-trained models available, ResNet50 and InceptionV3 have good performance on the imagenet dataset. Both these networks are insanely deep (50 layers and 48 layers, respectively). A resnet block consists of a convolutional layer, batch normalization, ReLU activation repeated three times and the output of this block is added with the input that was fed into the block. The weights of the ResNet blocks are completely frozen and are blocked from updating. The output of ResNet is connected to a stack of fully connected layers containing 50, 10 and 1 units respectively, with ReLU as their activation. The architecture of this model can be seen in Fig. 8.

The InceptionV3 is made up of several inception blocks. Each of these blocks consists of 4 separate networks that runs in parallel and finally concatenated into one. The general idea is to increase the depth of the network without affecting the memory and computational budget of the system. The filter size is kept at the maximum of 3x3 to extract maximum features and it uses the width increase technique at each layer to improve feature combination. Following a similar approach,we freeze the weights of the InceptionV3 blocks and the layers are blocked from updating. The output again is connected to a stack of fully connected layers containing 50, 10 and 1 units respectively. The architecture of this model can be seen in Fig. 9.

V. Experimentation

A. Algorithm

We trained all models using the following algorithm:

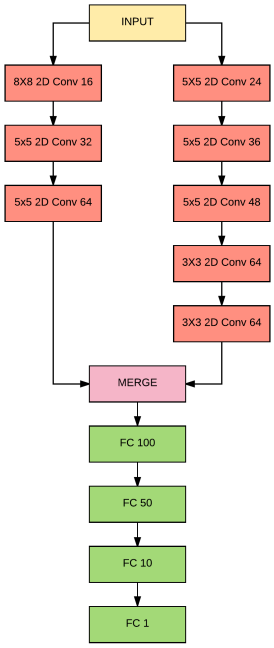


Fig. 7: Merged CNN Architecture

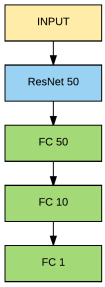


Fig. 8: ResNet 50 CNN Architecture

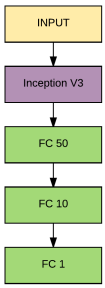


Fig. 9: Inception V3 CNN Architecture

1) Load the training data.
2) Normalize the image matrix from [0,255] to [0,1] and resize them if needed.
3) Split the data into train data, validation data and test data in the ratio of 0.6:0.2:0.2.
4) Input the model that we wish to train.
5) Compile the model with an appropriate optimizer and loss function.
6) Enter the number of epochs and batch size.
7) Fit the model batch-by-batch by using a suitable batch generator function.
8) Monitor the validation loss after each epoch and save the weights of model that has the least validation loss.
9) Terminate the training if the model seems to overfit the data (train loss $<<$ validation loss).
10) Load the weights of the model and evaluate the model on the test data.
11) Plot the results and verify the model output.

B. Results

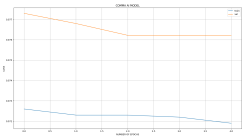
All these models were ran on the same datasets (training, validation and test). To establish a fair comparison we kept all the hyperparameters to be the same across all the different models. The number of epochs was set to be 5 and the batch size was taken to be 32. The results for each model is listed in Fig. 10. The result obtained from the comma AI model had a MSE of 0.0754 on the test data. And the MSE for the NVIDIA model was 0.0746 on the test data. We observe that the merged model outperformed the two baseline models with a MSE of 0.0312 on the test data. We also observe that ResNet50 and InceptionV3 models performed fairly well on the dataset. The MSE for ResNet50 and InceptionV3 were 0.0739 and 0.0733 respectively. The plot showing the training loss vs epochs for each model is displayed in Fig. 11.

	Training Set	**Validation Set**	**Test Set**
Comma AI Model	0.0719	0.07620	0.0754
NVIDIA Model	0.0726	0.07687	0.0746
Merged Model	0.0382	0.0324	0.0312
ResNet50 Model	0.0680	0.0732	0.0739
InceptionV3 Model	0.0720	0.07646	0.0733

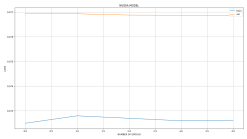
Fig. 10: MSE of all the Models

C. Conclusion

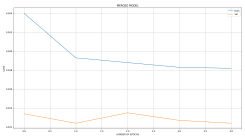
The test data loss was the least for the merged model. This justifies the hypothesis we made on our merged model. In future we plan to expand the network by having a larger and deeper network with LSTM units to extract and process the temporal information. Incorporating temporal information in a self driving car could play an important role. For instance, if the camera is fully saturated looking at the sun, knowing the information of the previous frames would allow us to make a better prediction. This however requires more computational resources and calls in for more memory but



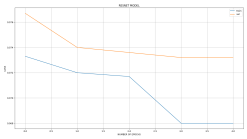
(a) Comma AI



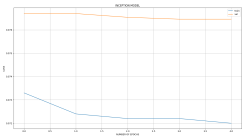
(b) NVIDIA Model



(c) Merged Model



(d) ResNet50 Model



(e) Inception Model

Fig. 11: Train Loss vs Validation Loss

with a right trade-off the performance of the current system could increased to several times.

We also observe that transfer learning method performed fairly well even though we had blocked all the layers from learning. This also suggests that transfer learning might yield better results if we had not blocked the entire network and allowed some part of the network to be trained on our dataset. Since, we had limited resources and time we couldn't afford to train the entire network and chose to go with the pre-trained weights. The future work lies on how to freeze the layers diligently and train the rest of the network such that we obtain better results without compromising on the memory and computational resources available to train the network.

REFERENCES

[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research 15 1929-1958*, 2014.

[2] Udacity, "self-driving car dataset," *https://github.com/udacity/self-drivingcar/tree/master/datasets*.

[3] M. Bojarski, D. Testa, D. Dworakowski, B. Firner, P. G. B. Flepp, L. D. Jackel, M. Monfort, U. Muller, and J. Zhang, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[4] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," *In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014.

[5] J. Deng, W. Dong, R. Socher, J. Li, K. Li, and F.-F. Li, "Imagenet."

[6] D. A. Pomerleau, "Alvinn, an autonomous land vehicle in a neural network," *Technical report, Carnegie Mellon University, Computer Science Department*, 1989.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *arXiv:1512.00567v3*, 2015.

[9] E. Santana and G. Hotz, "Learning a driving simulator," *arXiv:1608.01230v1*, 2016.

[10] S. Du, H. Guo, and A. Simpson, "Self-driving car steering angle prediction based on image recognition," *http://cs231n.stanford.edu/reports/2017/pdfs/626.pdf*, 2017.