Self-Driving Car Steering Angle Prediction Based on Image Recognition

Shuyang Du
shuyangd@stanford.edu

Haoli Guo
haoliguo@stanford.edu

Andrew Simpson
asimpso8@stanford.edu

Abstract

Self-driving vehicles have expanded dramatically over the last few years. Udacity has release a dataset containing, among other data, a set of images with the steering angle captured during driving. The Udacity challenge aimed to predict steering angle based on only the provided images.

We explore different models to perform high quality prediction of steering angles based on images using different deep learning techniques including Transfer Learning, 3D CNN, LSTM and ResNet. In addition to predicting steering angle, we also predict auxiliary variables like torque and speed to make the model know better about the driving condition and thus produce precise steering angle predictions.

1. Introduction

Self-driving vehicles are going to be of enormous economic impact over the coming decade. Creating models that meet or exceed the ability of a human driver could save thousands of lives a year. Udacity has an ongoing challenge to create an open source self-driving car. In their second challenge Udacity released a dataset of images taken while driving along with the corresponding steering angle and ancillary sensor data for a training set (left, right, and center cameras with interpolated angles based on camera angle). The goal of the challenge was to find a model that, given an image taken while driving, will minimize the RMSE (root mean square error) between what the model predicts and the actual steering angle produced by a human driver. In this project, we explore a variety of techniques including 3D convolutional neural networks, recurrent neural networks using LSTM, ResNets, etc. to output a predicted steering angle in numerical values.

The motivation of the project is to eliminate the need for hand-coding rules and instead create a system that learns how to drive by observing. Predicting steering angle is one important part of the end-to-end approach to self-driving car and would allow us to explore the full power of neural networks. For example, using only steering angle as the training signal, deep neural networks can automatically extract features to help position the road to make the prediction.

2. Related Work

Using a neural network for autonomous vehicle navigation was pioneered by Pomerleau (1989) [12] who built the Autonomous Land Vehicle in a Neural Network (ALVINN) system. The model structure was relatively simple, comprising a fully-connected network which is tiny by todays standard. The network predicted actions from pixel inputs applied to simple driving scenarios with few obstacles. However, it demonstrated the potential of neural networks for end-to-end autonomous navigation.

Last year, NVIDIA released a paper regarding a similar idea that benefited from ALVINN. In the paper [1], the authors used a relatively basic CNN architecture to extract features from the driving frames. The layout of the architecture can be seen in Figure 1. Augmentation of the data collected was found to be important. The authors used artificial shifts and rotations of the training set. Left and right cameras with interpolated steering angles were also incorporated. This framework was successful in relatively simple real-world scenarios, such as highway lane-following and driving in flat, obstacle-free courses.

Recently, more attempts on using deep CNNs and RNNs to tackle the challenges of video classification [8], scene parsing [4], and object detection [16] have stimulated the applications of more complicated CNN architectures in autonomous driving. ”Learning Spatiotemporal Features with 3D Convolutional Networks” introduces how to construct 3D Convolutional Networks to capture spatiotemporal features in a sequence of images or videos [17]. ”Beyond Short Snippets: Deep Networks for Video Classification” describes two ways including using LSTM to deal with videos [18]. ”Deep Residual Learning for Image Recognition” [6] and ”Densely Connected Convolutional Networks” [7] describe the techniques to construct residual connections between different layers and make it easier to train deep neural networks.

Besides of the CNN and/or RNN methods, there are more research initiatives applying deep learning techniques in autonomous driving challenges. Another line of work is to treat autonomous navigation as a video prediction task. Comma.ai [13] has proposed to learn a driving simulator with an approach that combines a Variational Auto-encoder

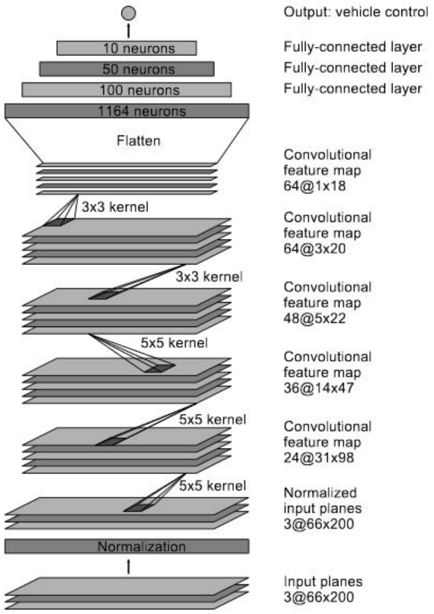

Figure 1. CNN architecture used in [1]. The network contains approximately 27 million connections and 250 thousand parameters.

(VAE) [10] and a Generative Adversarial Network (GAN) [5]. Their approach is able to keep predicting realistic-looking video for several frames based on previous frames despite the transition model being optimized without a cost function in the pixel space.

Moreover, deep reinforcement learning (RL) has also been applied to autonomous driving [3], [14]. RL has not been successful for automotive applications until some recent work shows the deep learning algorithms ability to learn good representations of the environment. This was demonstrated by learning of games like Atari and Go by Google DeepMind [11], [15]. Inspired by these work, [3] has proposed a framework for autonomous driving using deep RL. Their framework is extensible to include RNN for information integration, which enables the car to handle partially observable scenarios. The framework also integrates attention models, making use of the glimpse and action networks to direct the CNN kernels to the places of the input data that are relevant to the driving process.

3. Methods

We developed two types of models. The first one uses 3D convolutional layers. Since there is no pretrained 3D convolutional model available, the second one uses 2D pretrained convolutional layers from transfer learning.

3.1. 3D Convolutional Model with Residual Connections and Recurrent LSTM Layers

3.1.1 3D Convolutional Layer

How 3D convolutional layer works is similar to 2D conv, the only difference is that in addition to height and width, now we have the third dimension depth (temporal). Instead of having a 2D filter (if we ignore the channel dimension for a while) moving within the image along height and width, now we have a 3D filter moving along with height, width and depth. If the input has shape (D1, H1, W1, C), then the output would have shape (D2, H2, W2, F) where F is the number of filters. D2, H2, W2 could be calculated given stride and padding in its dimension.

3.1.2 Residual Connection

Since deep neural network is hard to train, we need residual connections here to help the training process. The idea of residual connection is to use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping. Without residual connection:
Fit $H(G(x))$ directly
With residual connection:
Fit $F(x) = H(G(x)) - G(x) - x$
After each 3D convolutional layers, we process its output through a fully connected layer and add it to the output of the whole Visual Feature extraction module.

3.1.3 Spatial Batch Normalization

Batch Normalization alleviates a lot of headaches with properly initializing neural networks by explicitly forcing the activations throughout a network to take on a unit gaussian distribution at the beginning of the training. Spatial batch normalization not only normalize among different samples but also among the spatial axis of images. Here we add spatial batch normalization after each 3D convolutional layer.

3.1.4 Recurrent Neural Networks and Long Short Term Memory

3.1.5 New Architecture

For self driving cars, incorporating temporal information could play an important role in production systems. For example, if the camera sensor is fully saturated looking at the sun, knowing the information of the previous frames would allow for a much better prediction than basing the steering angle prediction only on the saturated frame. As discussed earlier, 3D convolutional layers and recurrent layers incorporate temporal information. In this model we combined these two ways of using temporal information. We used the idea residual connection in constructing this model [6]. These connections allow for more of the gradient to pass through network by combining different layers. The model consisted five sequences of five frames of video shifted by one frame for the input (5x5x120x320x3). The values were selected to fit the computational budget. This allowed for both motion and differences in outputs between frames to be used. This model had 543,131 parameters. The architecture of the model can be seen in Figure 2.

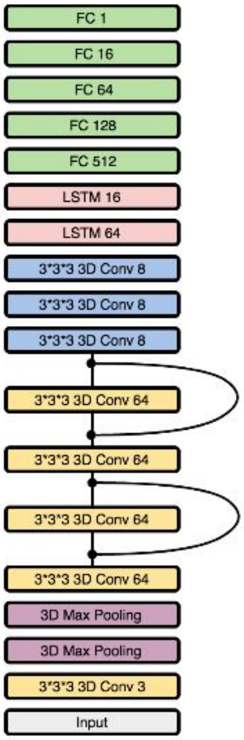

Figure 2. 3D convolutional model with residual connections and recurrent LSTM layers

The model consists of a few initial layers to shrink the size followed by ResNet like blocks of 3D convolutions with spatial batch normalization (only two of these in the trained model). Due to computational restraints, shrink layers were added to make the input to the LSTM layers much smaller. Only two levels of recurrent layers were used due to the speed of computation on these layers being much slower due to parts that must be done in a serial manner. The output of the recurrent layers was fed into a fully connected stack that ends with the angle prediction. All of these layers used rectified linear units, ReLUs, as their activation except the LSTM layers. Spatial batch normalization was used on the convolutional layers. The LSTM layers used tanh as their activation.

3.2. Transfer Learning

For this model, we used transfer learning. Transfer learning is a way of using high quality models that were trained on existing 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.

Of the pre-trained models available, ResNet50 had good performance for this dataset. This model was trained on ImageNet. The weights of the first 15 ResNet blocks were blocked from updating (first 45 individual layers out of 175 total). The output of ResNet50 was connected to a stack of fully connected layers containing 512, 256, 64, and 1 different units respectively. The architecture of this model can be seen in Figure 3 with the overall number of parameters being 24,784,641. The fully connected layers used ReLUs as their activation. The ResNet50 model consists of several different repeating blocks that form residual connections. The sizes of the filters vary from 64 to 512. A block is consistent of a convolutional layer, batch normalization, ReLU activation repeated three times and the input layer output combined with the last layer.

Other sizes of locking were attempted, but produced either poor results or were slow in training. For example, training only the last 5 blocks provided poor results, which were only slightly better than predicting a steering angle of zero for all inputs. The model took as input images of 224x224x3. The only augmentation provided for this model was mirrored images. Due to the size constraints of the input into ResNet50, cropping was not used as it involved stretching the image. The filters in the pretrained model were not trained on stretched images, so the filters may not activate as well on the stretched data (RMSE of 0.0891 on the validation set after 32 epochs). Additionally, using the left and the right cameras from the training set proved not to be useful for the 32 epochs used to train (0.17 RMSE on the validation set).

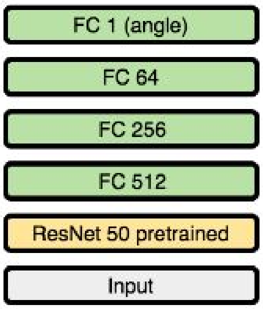

Figure 3. Architecture used for transfer learning model.

4. Dataset and Features

The dataset we used is provided by Udacity, which is generated by NVIDIAs DAVE-2 System [1]. Specifically, 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 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 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 data set contains 101397 frames and corresponding labels including steering angle, torque and speed. We further split this data set into training and validation in a 80/20 fashion. And there is also a test set which contains 5615 frames. The original resolution of the image is 640x480.

Training images come from 5 different driving videos:

1. 221 seconds, direct sunlight, many lighting changes. Good turns in beginning, discontinuous shoulder lines, ends in lane merge, divided highway
2. 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 can't 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.
3. 99 seconds, divided highway segment of return trip over the summit
4. 212 seconds, guardrail and two lane road, shadows in beginning may make training difficult, mostly normalizes towards the end
5. 371 seconds, divided multi-lane highway with a fair amount of traffic

Figure 4 shows typical images for different light, traffic and driving conditions.

4.1. Data Augmentation Methods

4.1.1 Brightness Augmentation

We randomly change the brightness to simulate different light conditions. We generate augmented images with different brightness by first converting images to HSV, scaling up or down the V channel and converting back to the RGB channel. Following are typical augmented images.

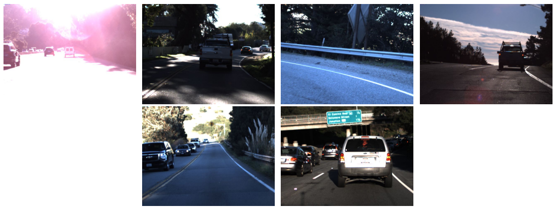

Figure 4. Example images from the dataset. From left to right, bight sun, shadows, sharp left turn, up hill, straight, and heavy traffic conditions.

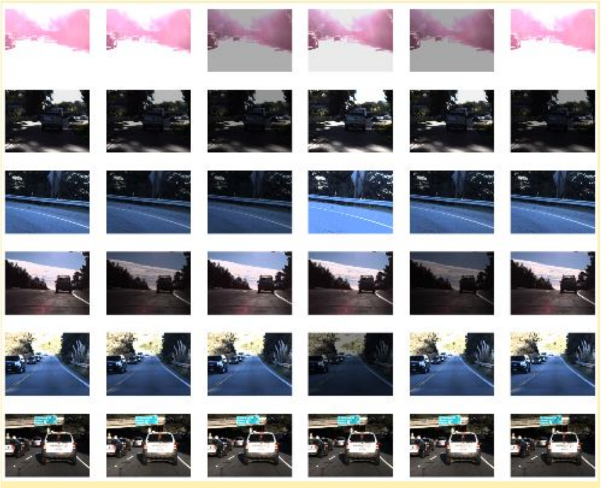

Figure 5. Brightness augmentation examples

4.1.2 Shadow Augmentation

We also cast random shadows across images. The intuition is that even the camera has been shadowed (maybe by rainfall or dust), the model is still expected to predict the correct steering angle. This is implemented by choosing random points and shading all points on one side of the image.

4.1.3 Horizontal and Vertical Shifts

We will shift the camera images horizontally to simulate the effect of car being at different positions on the road, and add an offset corresponding to the shift to the steering angle. We will also shift the images vertically by a random number to simulate the effect of driving up or down the slope.

4.2. Preprocessing

For each image, we normalize the value range from [0, 255] to [-1, 1] by normalizing by image=-1+2*original images/255. We further rescale the image to a 224x224x3

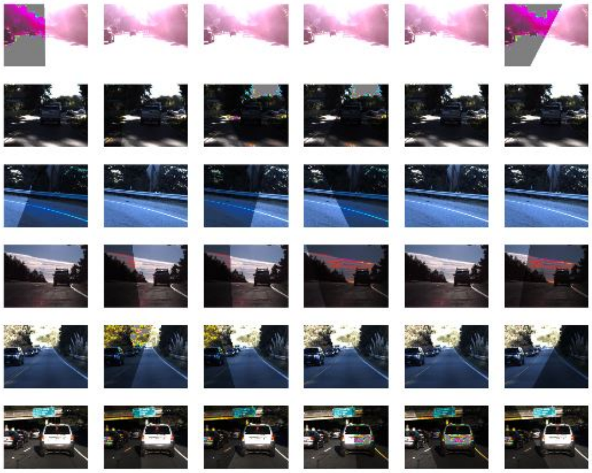

Figure 6. Shadow augmentation examples

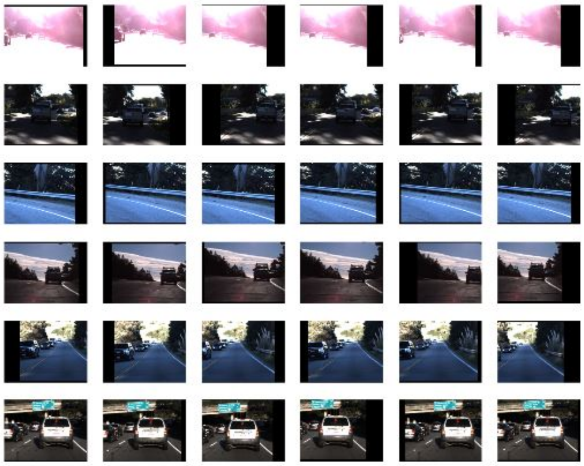

Figure 7. Shift augmentation examples

square image for transfer learning model. For the 3D LSTM model we crop the sky out of the image to produce data the size of 120x320x3.

5. Experiments, Results, and Discussion

5.1. Data Augmentation

In order to establish a baseline for our research, we used the architecture from [1] to test different forms of data augmentation. Teams in the Udacity challenge noted that data augmentation was helpful along with the original NVIDIA researchers. Knowing which forms of augmentation work well for the amount of time and computational available would be helpful in training our new models. The NVIDIA model architecture seen previously in Figure 1. The input to this model was 120x320x3 with a batch size of 32. In the NVIDIA paper [1] it was not clear how they optimized the loss function. For this experiment, we used the default parameters of Adam (see [9]) provided in Keras (learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e - 8$, and $decay = 0$).

Three different levels of augmentation were examined. The first had minimal augmentation with only using random flips and cropping of the top of the image. Randomly flipping the input images eliminates the bias towards right turns found in the dataset. Cropping the top of the image eliminates the sky from the image, which should not play a role in how to turn predict the steering angle. A second form of augmentation had the same augmentation as the minimal version along with small rotations (-5 to +5 degrees), shifts (25 pixels), and small brightness changes of the image. The final Heavier version of augmentation used more exaggerated effects of the second version including large angle rotations (up to 30 degrees), large shadows, shifts, and larger brightness changes were used. Results from this experiment can be see in Table 1.

Table 1. RMSE on the validation set using the NVIDIA architecture for different levels of data augmentation with 32 epochs.

Minimal	**Moderate**	**Heavy**
0.09	0.10	0.19

Using heavy data augmentation produced very poor results that were not much above predicting a steering angle of 0 for all the data. The moderate augmentation produced good results; however the minimal augmentation produced the best results. These results could be explained by only training for 32 epochs. Heavy augmentation could be hard for the model to pick up on such drastic shifts. Similarly, the moderate version may have outperformed the minimal version over more epochs. In a later section visualization of these tests will be examined. For our new models, we chose to use minimal augmentation.

5.2. Training Process

5.2.1 Loss Function and Optimization

For all models used, the mean-square-loss function was used. This loss function is common for regression problems. The MSE punishes large deviations harshly. This function is simple the mean of the sum of the squared differences between the actual and predicted results (see Equation 1). The scores for the Udacity challenge were reported as the root-mean-square-error, RMSE, which is simply the square root of the MSE.

$$\begin{aligned} MSE &= \frac{1}{n}\sum(y_i - \hat{y_i})^2 \\ RMSE &= \sqrt{\frac{1}{n}\sum(y_i - \hat{y_i})^2} \end{aligned} \tag{1}$$

To optimize this loss, the Adam optimizer was used [9]. This optimizers is often the go to choice for deep learning application. This optimization method usually substantially outperforms more generic stochastic gradient decent methods. Initial testing of these models indicate that their loss levels get stuck after a few epochs. The decay rate of the optimizer was updated from 0 to the learning rate divided by the number of epochs. The other default values of the Keras Adam optimizer showed good results during training (learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e - 8$, and decay=learning rate/batch size).

5.3. Feature Visualization

In order to examine what our networks find relevant in an image, we can use saliency maps. These maps can show how the gradient flows back to the image highlighting the most salient areas. A similar approach was used in a recent NVIDIA paper [2].

5.3.1 Data Augmentation Experiment

What these models found important can be visualized in Figure 8. The minimal model found the lane markers important. In the moderate model more of the lane markers were found to be important; however, this model's saliency maps appeared more noisy, which could explain its slightly decreased performance. In the heavy model almost no areas were found to be salient, which is understandable due to its poor performance.

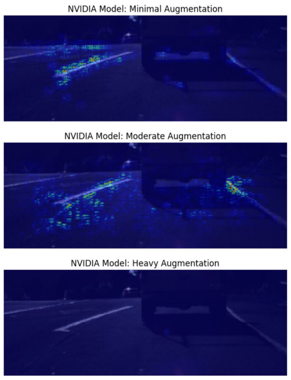
NVIDIA Model: Minimal Augmentation
NVIDIA Model: Moderate Augmentation
NVIDIA Model: Heavy Augmentation

Figure 8. NVIDIA model saliency maps for different levels of augmentation.

5.3.2 3D Convolutional LSTM Model

This model produced interesting saliency maps. In examining on of the video clips fed into the model, we can see that the salient features change frame to frame in Figure 9. The salient features seem to change from frame to frame, which would indicate that the changes between frames are important.

This sequence of frames can be collapse into a single image, which is shown in Figure 10. The collapsed version helps to visualize this better. The expressed salient features do cluster around road markers, but they also cluster around other vehicles and their shadows. This model may be using information about the car in front in order to make a steering angle prediction along with the road markers.

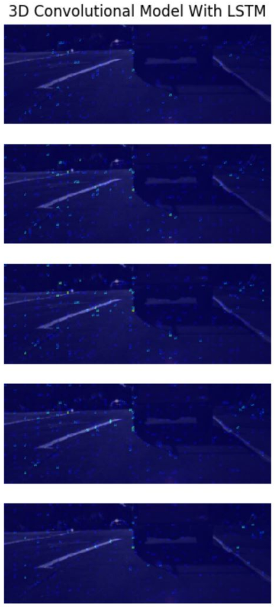

Figure 9. Saliency map for a sequence in the 3D convolutional LSTM model for an example image.

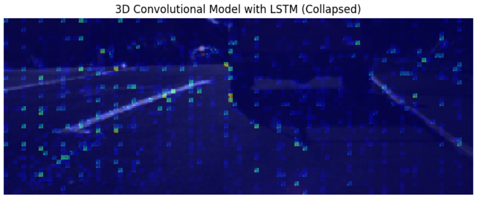

Figure 10. Saliency map for the 3D convolutional LSTM model for an example image.

5.3.3 Transfer Learning Model

An example saliency map for the ResNet50 transfer learning model can be seen in Figure 11. The model does appear to have salient features on the road markers; however, there are also regularly spaced blotches. These blotches may be artifacts from using this type pretrained model with residual connections. Although this model had the best overall results, its saliency maps did not match well with the expectation of what would be expected for salient features in predicting steering angles from road images.

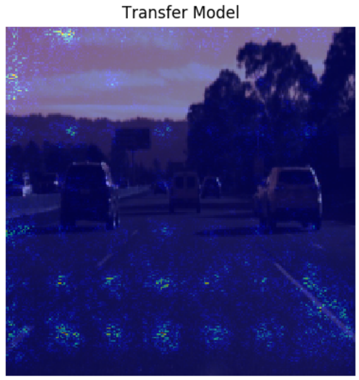

Figure 11. Transfer learning model (ResNet50) saliency map for an example image.

5.4. Results

These models were all ran on the same datasets. The results for each model is listed in Table 2. The results from the 3D convolutional model with LSTM and residual connections had a RMSE for the test set of 0.1123. Since the Udacity challenge is over, the results can be compared to the leader board. For the 3D LSTM model, the results on the test set would have put in in tenth place overall. The ResNet50 transfer model had the best results overall with a RMSE of 0.0709 on the test set. This result would have placed the model in fourth place overall in the challenge. This is without using any external functions for the models (some teams used an external smoothing function in conjunction with their deep learning models).

Table 2. RMSE for the models on the Udacity dataset.

	Training Set	Validation Set	Test Set
Predict 0	0.2716	0.2130	0.2076
3D LSTM	0.0539	0.1139	0.1123
Transfer	0.0212	0.0775	0.0709
NVIDIA	0.0750	0.0995	0.0986

In order to help visualization of what this level of error looks like, an example overlay for random images is seen in Figure 12. The green circle indicates the true angle and the red circle indicates the predicted angle. The predictions are generated from the ResNet50 transfer learning model.

5.5. Discussion

For the amount of epochs we used, only minimal data augmentation proved to be of any major use for these model. For more expansive training, the strategy of data augmentation can allow for near infinite training data given the right strategy.

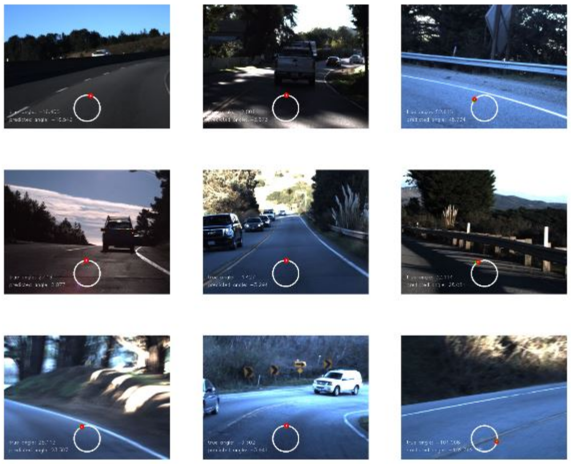

Figure 12. Example actual vs. predicted angle on unprocessed images (transfer model).

Overall, these models showed that they were competitive with other top models from the Udacity challenge. For the 3D LSTM model, with more time and computational resources, this model could have been expanded to take in a longer period of video along with more ResNet blocks. Expanding the model in this way could have produced superior results. One of the teams near the top of the competition used a full 250 frames or 2.5 seconds of video.

For the Resnet50 transfer model, the strategy of using a pre-trained model, locking approximately the first quarter layers, training the deeper layers with the existing weights, and connecting to a fully connected stack proved to be effective in producing a high quality and competitive model for the Udacity self-driving car dataset. It was surprising that this model outperform the other model. The architecture of this model takes no temporal data, yet it still predicts very good values.

Both of these models appeared to have had some overfitting with the ResNet50 model having more of an issue with this. Data augmentation could act as a form of regularization for this model. Different teams in the Udacity challenge have tried different regularization method including dropout and L_2 regularization. The results for using this regularization methods was mixed with some teams claiming good results and others having less success.

6. Conclusion and Future Work

In examining the final leader board from Udacity our models would have placed fourth (transfer learning model) and tenth (3D convolutional model with LSTM layers). These results were produced solely from the models without any external smoothing function. We have shown that both transfer learning and a more advanced architecture have promise in the field of autonomous vehicles. The 3D model was limited by computational resources, but overall it still provided a good result from a novel architecture. In future work the 3D model's architecture could be expanded by having a larger and deeper layers, which may produce better results.

These models are far from perfect and there is substantial research that still needs to be done on the subject before models like these can be deployed widely to transport the public. These models may benefit from a wider range of training data. For a production system, a model would have to be able to handle the environment in snowy conditions. Generate adversarial models, GANs, could be used to transform a summer training set into a winter one. Additionally, GANs could be used to generate more scenes with sharp angles. Additionally, a high quality simulator could be used with deep reinforcement learning. A potential reward function could be getting from one point to another while minimizing time, maximizing smoothness of the ride, staying in the correct lane/following the rules of the road, and not hitting objects.

References

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*, 2017.

[3] A. El Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *Autonomous Vehicles and Machines, Electronic Imaging*, 2017.

[4] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2013.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[7] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[8] 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*, pages 1725–1732, 2014.

[9] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

[13] E. Santana and G. Hotz. Learning a driving simulator. *arXiv preprint arXiv:1608.01230*, 2016.

[14] S. Shalev-Shwartz, S. Shammah, and A. Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[15] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[16] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pages 2553–2561, 2013.

[17] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[18] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015.