Deeper Direct Perception in Autonomous Driving

Zuozhen Liu
Stanford University
450 Serra Mall, Stanford, CA 94305
zliu2@stanford.edu

Yixin Wang
Stanford University
450 Serra Mall, Stanford, CA 94305
wyixin@stanford.edu

Abstract

Deep neural network architectures[14][4] have achieved amazing results on classification tasks in the ImageNet challenge (ILSVRC)[13] over recent years. While these networks are optimized for object classification problem with a given set of target classes, they also demonstrate strong capability in generalizing to other vision tasks such as object detection[12], pose estimation[17], etc. In this paper, we propose to apply transfer learning to the domain of autonomous driving based on a direct perception approach[2]. We present and evaluate our methods in fine-tuning pre-trained convolutional neural netowrk(CNN) models[14] for a regression task. Finally, we visualize and analyse the results of our model on test set as well as in game.

1. Introduction

Autonomous driving has been an active area of AI research for over decades. Recent successes in Computer Vision over the past decade have significantly contributed to the development of modern autonomous driving systems such as Google self-driving cars. While state-of-the-art systems are already capable of navigation in different road conditions and also exhibit robustness to different traffic scenarios, these systems are still very complex and require costly devices such as laser sensors and radars. These devices help constantly scan the surroundings with fine-grained details but often times these details contribute very little to driving command decisions and would simply add noise to the system.

In an attempt to resolve this problem, a different approach called direct perception is proposed in [2]. The idea is to make the system extract a few key relevant indicators instead of a holistic 3D model of the environment. The goal is to come up with a condensed representation such that it encodes enough information for a logic controller to infer the orientation of the host car as well as relevant cars in the traffic and then make corresponding decisions.

In this project, we focus on applying transfer learning to the direct perception approach[2]. Specifically, given an input image of first-person front view during driving, the system relies on a feature mapping to output a number of key perception indicators. These real value indicators are directly related to the affordance of a road/traffic state for driving and can be fed into a logic controller to output the final driving commands. The mapping process in the pipeline can be learned via many methods. Since Convolutional Neural Network (CNN) hav proven its power in learning effective feature representations of complex scenes from raw images, a natural idea is to transfer the learning of CNN to this regression task in order to map an input image to real-value key perception indicators.

2. Related Work

The majority of autonomous driving systems can be categorized into three major paradigms shown in Figure 1: mediated perception approaches, behavior reflex approaches and direct perception approaches.

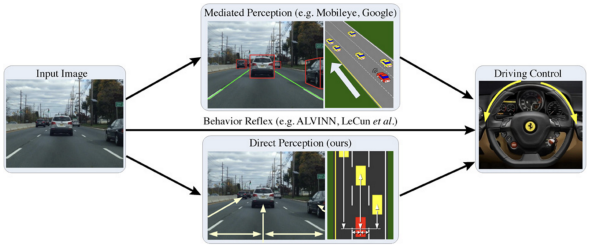
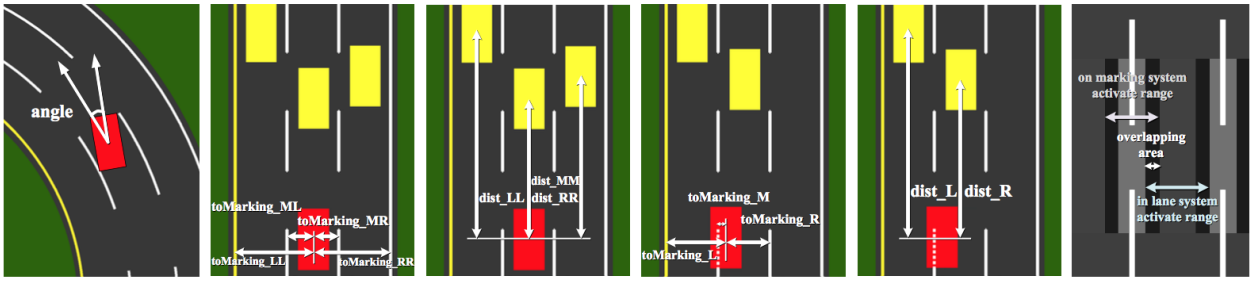


Figure 1. Illustration of three paradigms[2]

Mediated perception approaches essentially rely on independent modules[3] that are optimized for different specialized recognition tasks including lane detection[1], car detection[5][9], etc. The outputs of these modules are then integrated into a unified world representation from which the final driving decisions are made. Even though most state-of-the-art systems[10] fall into this category, the complexity and cost concerns still remain a major challenge.

Behavior reflex approaches[11] are drastically different



(a) angle (b) in lane: toMarking (c) in lane: dist (d) on mark.: toMarking (e) on marking: dist (f) overlapping area

Figure 2. Illustration of the meaning of affordance indicators [2]

from the approaches mentioned above in that the sensor inputs are directly applied to learning driving actions. While this idea is very simple and has been proven to handle lane navigation quite well, it lacks high dimensional representations to deal with complicated traffic scenarios.

Direct Perception approach is a new paradigm introduced in [2]. The paper also adopts CNN to learn a mapping from input image to key perception indicators and shows the host car is able to navigate itself in simulated highway scenario in TORCS[18]. While this idea is innovative, the paper follows an AlexNet architecture[8] which is superseded by deeper CNN architectures in ILSVRC. In this project, we will instead use pre-trained VGGnet[14] from Caffe[6] to learn a better estimation of perception indicators from raw image inputs.

3. Problem Statement

3.1. Direct Perception

The input into our CNN is a sequence of 280 x 210 RGB image frames representing first-person driving view in TORCS. We map the input images to 13 key perception indicators shown in Figure 2, namely *angle*, *toMarking_LL*, *toMarking_ML*, *toMarking_MR*, *toMarking_RR*, *dist_LL*, *dist_MM*, *dist_RR*, *toMarking_L*, *toMarking_M*, *toMarking_R*, *dist_L*, *dist_R*. The semantic meaning of these indicators are listed in Table 1.

3.2. Controller Logic

Next step is to implement a logic controller that takes these 13 key perception indicators as well as the speed of the car to output driving actions. In testing phase, these actions are sent back to TORCS at a frequency of 10 Hz to control the host car on the fly. We will be using the same controller as described in [2]. At a high-level, the controller navigates the host car in lane and avoids potential collision by either switching to open lanes or slowing down. The details of the controller is illustrated in Figure 3.

1) angle: angle between the cars heading and the tangent of the road
in lane system, when driving in the lane:
2) toMarking_LL: distance to the left lane marking of the left lane
3) toMarking_ML: distance to the left lane marking of the current lane
4) toMarking_MR: distance to the right lane marking of the current lane
5) toMarking_RR: distance to the right lane marking of the right lane
6) dist_LL: distance to the preceding car in the left lane
7) dist_MM: distance to the preceding car in the current lane
8) dist_RR: distance to the preceding car in the right lane
on marking system, when driving on the lane marking:
9) toMarking_L: distance to the left lane marking
10) toMarking_M: distance to the central lane marking
11) toMarking_R: distance to the right lane marking
12) dist_L: distance to the preceding car in the left lane
13) dist_R: distance to the preceding car in the right lane

Table 1. Key indicator names and meanings

while (in autonomous driving mode)
  ConvNet outputs affordance indicators
  check availability of both the left and right lanes
  if (approaching the preceding car in the same lane)
    if (left lane exists **and** available **and** lane changing allowable)
      left lane changing decision made
    else if (right lane exists **and** available **and** lane changing allowable)
      right lane changing decision made
    else
      slow down decision made
  if (normal driving)
    $center\_line$= center line of current lane
  else if (left/right lane changing)
    $center\_line$= center line of objective lane
  compute steering command
  compute $desired\_speed$
  compute acceleration/brake command based on $desired\_speed$

Figure 3. Controller Logic [2]

4. Technical Approach

4.1. VGG Network

VGGNet has achieving amazing performance in ILSVRC 2014 by exploring a much deeper network architecture. Even though VGGNet has a slightly weaker classification accuracy in ILSVRC than the winner, GoogleNet[15], it actually outperforms GoogleNet in multiple transfer learning tasks. Therefore, we decide to fine-

tune VGGnet for the regression task on autonomous driving.

The key difference between VGGnet and other network architectures is the use of small size convolution filters. The author argued that using a stack of three 3x3 filters has the same receptive field as a 7x7 filter. However, the total number of weight parameters, given C channels from the input, amounts to $27C^2$ instead of $49C^2$, resulting in 45% less parameters. As a result, VGGnet architecture applies homogeneous 3x3 convolution filters with a stride of 2 and 2x2 pooling filters with a stride of 2 that reduces the data dimension by half. The number of filters are [64, 128, 256, 512, 512] which correspond to all convolution filters before each pooling layer.

Specifically, we choose VGG-16 pre-trained model on ImageNet from Caffe. The input image dimension is now 280 x 210 and the pre-trained VGGnet would break during forward pass when the input propagates to the fully connected layers. Additionally, the original network uses Softmax loss for object classification which does not work for regression tasks. Our proposal is to reinitialize the entire fully connected layers as [4096 - 256 - 13] and attach a regression head with Euclidean loss. The detailed architecture is displayed in Figure 4.

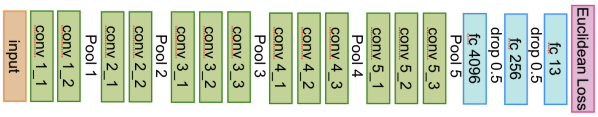


Figure 4. Modified VGG-16 architecture

4.2. Euclidean Loss

Given prediction and groud truth labels for m examples, $\hat{y}_i, y_i \in \mathbb{R}^n, i \in [1, m]$, the Euclidean loss or L2 loss is defined as:

$$L = \frac{1}{2m} \sum_{i=1}^{m} ||\hat{y}_i - y_i||_2^2$$

As the real-value indicators have rather different value ranges, we normalize the outputs to [0.1, 0.9] before we calculate the Euclidean loss. Given that Euclidean loss is much harder to optimize than a more stable loss such as Softmax, as is stated in CS231n class notes, we experience some difficulty with tuning the hyper-parameters in the networks. Detailed analysis on hyper-parameters can be found in 6.1.

4.3. Optimization

Our VGG model is optimized using Stochastic Gradient Descent(SGD) with momentum update. Given momentum μ, learning rate α, the update rule is the following:

$$\begin{aligned} v &= \mu \times v - \alpha \times dw \\ w &= w + v \end{aligned} \tag{1}$$

v represents the momentum which accumulates a history of previous weight updates. This method has a faster convergence than vanilla SGD but is slower than Adam[7] or RMSProp[16]. Unfortunately, we are constrained to work with an older version of Caffe developed in [2] to enable interface support with TORCS. Therefore, more advanced update methods are not supported which to some degree limites the performance of our models.

5. Dataset and Framework

5.1. Dataset

We use the TORCS dataset collected by [2], which are extracted from a 12-hour video recording of a human player driving a host car in TORCS. The original dataset includes a training set and a test set stored in levelDB format. However, we find that the images in the test set are corrupted (purely noise and nothing else). Since both the training and test set are collected in exactly the same way, we decide to split the training set into three subsets, namely training, validation and test. We have 150,000 images for training, 10,000 images for validation, and 10,000 images for testing.

Each sample in the dataset consists of an image and corresponding ground truth label. Figure 5 shows two example images from our dataset. Each image is the first-person view of the current traffic condition. The ground truth label consists of 13 float number fields, corresponding to the 13 indicators that we try to predict. The ground truth for indicators are directly collected from the game engine (note that the TORCS game itself needs these values to render scenes). Thus part of the source code for TORCS is modified to take out these ground truth, and a modified version of TORCS is compiled.

Also, note that apart from the host car, there are also 12 AI cars to create some traffic scenes in the dataset.

Data pre-processing We perform mean subtraction on both training and test images. Each channel is subtracted with its mean value to make data zero-centered, thus allowing for better training. The mean value file is provided by TORCS dataset.

There are two test modes. One is to evaluate the CNN models on the test set, the other is to evaluate the networks in real-time against TORCS. In the first case, test images are fixed so that we can compare the performance of different methods using the same test set. In the second case, test images are generated on-the-fly, namely, by taking screenshots of the TORCS game and feeding these screenshots into the neural network via shared memory. This mode will be discussed in detail in 6.6. We will report the results based on test set, and we will include the link to a video demo for real-time driving control in the TORCS game.

Figure 5. the TORCS Dataset image samples

5.2. Framework

Our development is based on the Caffe deep learning framework [6]. However, the Euclidean loss layer is modified to enable normalization of the loss of 13 output indicators within range 0.1 to 0.9. The normalization process requires prior knowledge, e.g. the width of the lane in the TORCS game, etc., and therefore such a layer modification is necessary.

6. Experiments and Results

We use the CNN in [2], which is is a standard AlexNet architecture[8], as our baseline. In this project, we propose to use VGG-16[14], which is a deeper network than baseline, and we fine-tune the final 3 fully connected layers [4096 - 256 - 13] to adapt to our problem. The final loss layer is Euclidean loss.

We train and test both the baseline model and our model (we will refer to CNN model in [2] as "baseline", and our model as "VGG" later in this report). In this section, we report the training loss, training time, test performance and detailed analysis. The detailed description of our modified VGG network is already elaborated in 4.1. Therefore, here we only briefly describe the baseline AlexNet model. The first 5 layers are standard [conv - relu - pool] layers, followed by 4 fully connected layers [4096 - 4096 - 256 - 13]. The loss layer is Euclidean loss. We trained the baseline model using exactly the same hyper-parameters as described in [2], from scratch instead of fine-tuned.

In the experiment, we re-train the baseline AlexNet model because
(1) Chen's experiment results as reported in [2] is based on a different test set which is found to be corrupted. To ensure that both two models run on the same test set, we cannot use the performance reported in [2].
(2) Nor can we use Chen's pre-trained model because we partitioned the training set into train/val/test, as described in 5.1. Chen's pre-trained model is trained on the entire training set, and therefore must have seen our "test" images.

For our VGG model, we fine-tune the network by fixing the pre-trained weights for all convolutional layers and only updating the weights for fully connected layers. We perform training and testing on a single NVIDIA GTX TITAN X GPU. Training time is measured and reported based on this platform.

6.1. Hyper-parameters in Training

Learning Rate We first start by tuning the learning rate for our network on validation set. Since we initialize the network with pre-trained weights, the learning rate should be much smaller than 1e-2 which is a common practice as recommended in Caffe. Therefore, we experiment with base learning rate in the range of [1e-3, 1e-4, 1e-5] for the fully connected layers for a small number of iterations. From Figure 6, we observe an effective learning rate of 1e-4, corresponding to the middle figure, yields the best result.

When training baseline from scratch, we use $\mathrm{lr} = 1e-2$, and learning rate decay $gamma = 0.9, step = 1000$, i.e., learning rate is multiplied by 0.9 every 1000 iterations. We choose a large learning rate because the baseline model is trained from scratch, and thus needs a relatively large initial learning rate to help it converge faster and escape local minima.

When fine-tuning VGG model, we use $\mathrm{lr} = 1e-4$, and learning rate decay $gamma = 0.9, step = 1000$. Smaller learning rates are usually preferred by fine-tuning tasks, because the situation is equivalent to the close-to-converge state of the network.

Dropout We decided not to use dropout in training. We experimented with dropout parameters, but it turned out that dropout is making loss decrease very unstable. Moreover, because our batch size is small due to memory limitations, which leads to instability by nature, applying dropout is not good for convergence. This verifies the statement on lecture notes that dropout might not be a good idea for regression problem.

Momentum We use momentum parameter $\mu = 0.9$. This parameter is usually set to 0.9 as a common practice. In our experiment, we tried turning off the momentum which led to a much slower convergence as explained in 4.3.

6.2. Error Metric

The error metric we use for test is mean abstract error (MAE) between ground truth and estimated values for indicators.

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{13}\sum_{i=0}^{12}|\hat{y_i} - y_i|\right),$$

where N is batch size, $y_i (i \in [0..12])$ is the ground truth of 13 indicators and $\hat{y}_i (i \in [0..12])$ are their corresponding estimated value.

6.3. Training

Both systems are trained on 15,0000 samples, 1 epoch and tested on 10,000 samples. Baseline system is trained for

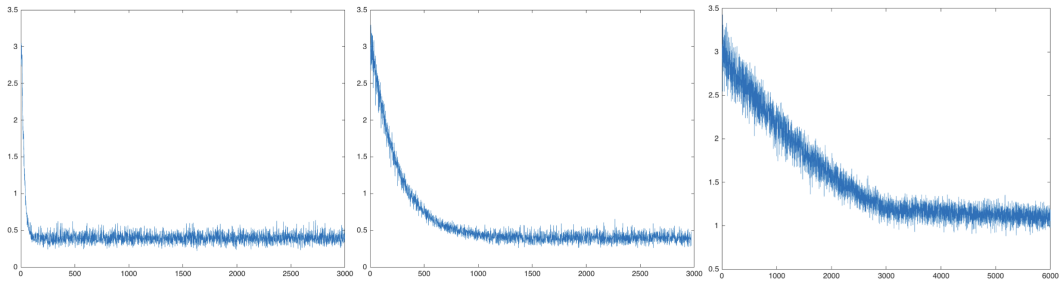


Figure 6. Loss curve for learning rate of 1e-3, 1e-4, 1e-5 (left to right)

10,000 iterations with batch size of 15. VGG is trained for 15,000 iterations with batch size of 10. Batch size is limited because of memory limitation on GPU. Limited batch size also leads to larger fluctuation in training loss.

Figure 7 and Figure 8 illustrates regression loss decrease against number of iterations for baseline and VGG network, respectively. The baseline model loss encounters a plateau around iteration 100 to 500, and then begin to linearly decrease, and nearly converges at around 0.13. VGG model loss decreases very quickly at first, and later linearly as learning rate decays, though the slope is very small. It nearly converges at around 0.17. The loss of both models are still linearly decreasing when we cut off the training process. However, we were not able to train for larger epochs due to time limitation.

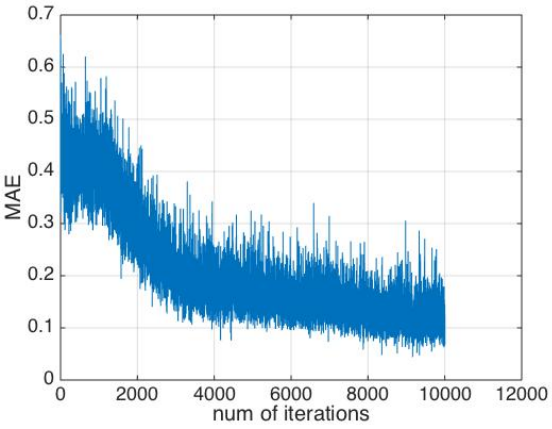


Figure 7. Training loss of baseline network

Sigmoid layer before output The model proposed in [2] attached a sigmoid layer between the final fully-connected layer and Euclidean loss layer to normalize loss to range [0.1,0.9]. However, when we were training, we found that adding sigmoid layer leads to slower convergence. This might be because removing the sigmoid layer allows for outputs with larger discrepancy to true value to learn faster. On the other hand, if a sigmoid layer is attached, outputs that are wildly off might actually learn more slowly because of the saturation of sigmoid layer. After 10,000 iterations,

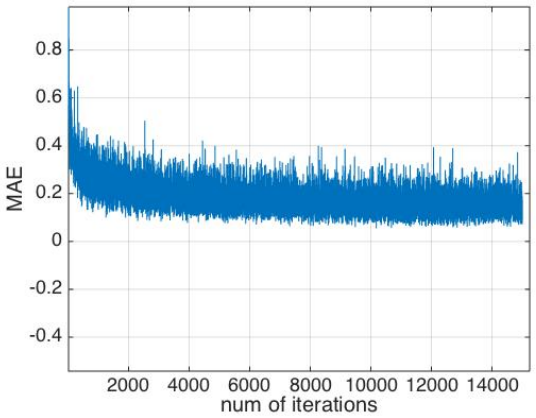


Figure 8. Training loss of VGG

the loss decreases to 0.24 if we use a sigmoid layer, while it decreases to 0.20 if we remove this layer.

Training time Fine tuning the VGG network is much faster than training VGG from scratch, mainly because VGG architecture is very deep. Training AlexNet from scratch, however, is an acceptable approach for us because the shallower structure of AlexNet means each iteration in training will take shorter time. The time for 1 iteration in training is reported in Table 2.

Target	Elapsed Time (s)
Train AlexNet from scratch	0.52
Train VGG from scratch	0.81
Fine-Tune 3 fc layers in VGG	0.44

Table 2. Time for 1 iteration in training

6.4. Results

We test the baseline model and VGG model on the 10,000 test images. The MAE of both models are shown in Table 3. We also included the performance of directed perception model with GIST descriptors instead of CNN features[2] for comparison.

Our VGG network is performing much better than the GIST based system. However, it is not performing so well

Params	angle	to_LL	to_ML	to_MR	to_RR	dist_LL	dist_MM	dist_RR	to_L	to_M	to_R	dist_L	dist_R
GIST	0.051	1.033	0.596	0.598	1.140	18.561	13.081	20.542	1.201	1.310	1.462	30.164	30.138
AlexNet	0.048	0.221	0.203	0.199	0.223	11.574	9.842	12.319	0.174	0.195	0.178	10.920	8.927
VGG	0.062	0.569	0.466	0.431	0.568	13.526	10.901	13.627	0.577	0.575	0.555	14.572	12.266

Table 3. Performance of GIST, baseline AlexNet, VGG based system

as the re-trained baseline AlexNet system in [2]. Because we are freezing the convolution layers and only fine-tuning the final 3 fully-connected layers of VGG, the performance might be constrained. If we had had time and computational resources to run more iterations, we would not have to freeze the convolution layers, and allow gradients to propagate back to previous layers to fine-tune convolution parameters. This might lead to better performance. Anyhow, our VGG network is achieving competitive results that are comparable to the results in [2].

6.5. Error Analysis

When we evaluate our VGG model on the test set, we also visualize the estimation outputs together with ground truth to generate qualitative results. We present three test samples below to discuss about the performance of our networks.

In each sample, the visualization is displayed on the left while the test image is shown on the right. In the visualization, solid boxes represent ground truth car locations. The host car is marked in red and the AI cars are marked in yellow. Green bounding box is the estimate of host car based on a number of toMarking indicators. The purple bounding boxes are estimates of AI cars using distance indicators.

In Figure 9, the host car is driving in lane as shown in (c) from Figure 2. In this case, the host car estimates its distance from cars, if present, in all three lanes. We can observe that the bounding box for the host car is very accurate and this stays true for a vast majority of in-lane situations. Thus, the VGG model is able to navigate itself along the racing tracks quite well (without disturbance from traffics).

Figure 9. Test sample, in lane

In Figure 10, the car is in an on-marking configuration as shown in (e) from Figure 2. In this case, only the onMarking indicators are active for the host car. Therefore, it only effectively "observes" the two lanes it's currently driving in-between. We observe that the model is less robust to this type of configuration which usually happens when the car is switching lanes. In this example, the estimates are relatively good.

Figure 10. Test sample, on marking

In Figure 11, we can clearly categorize this sample as one of the failure scenarios where the estimates are completely off from ground truth. The cause is that the model "thinks" the host car is in between first and second lanes which is off by one lane from the ground truth. We believe that this is a fundamental limitation of the perception indicators for onMarking scenarios in that these indicators fail to grasp a global understanding of the position of the host car with respect to entire track, which usually has more than just two lanes.

A simple improvement is to add in one more indicator representing the number of lanes to the left of the host car. The indicator would help the model distinguish Figure 10 from Figure 11, e.g, values are 1, 2 respectively. However, it still requires further investigation to validate the performance impact.

Figure 11. Test sample, on marking error

6.6. Real time control in TORCS game

The network can also run in real-time in collaboration with the TORCS game, and the pipeline is described as fol-

lows.

The system is composed of several components, namely a TORCS simulator, a deep network (e.g. AlexNet or VGG) and a Driving Controller. The relationship between these three major components are shown in Figure 12. Specifically, for the architecture of deep network, we use VGG [14]. The TORCS simulator and VGG use shared memory to enable the passing of images generated in real-time. The pipeline is shown below.

(1) The TORCS game engine writes the rendered image to shared memory in real-time.

(2) VGG reads image from shared memory, and does a forward computation, mapping these images to a set of 13 key indicators.

(3) These 13 key indicators are then written to shared memory.

(4) The Driving Controller consists of a set of hard-coded control logic. It reads key indicators from shared memory, and maps these 13 key indicators to a set of control actions.

(5) The Driving Controller writes control actions to shared memory.

(6) TORCS games engine reads these control actions from shared memory, and manipulates the host car in TORCS simulator.

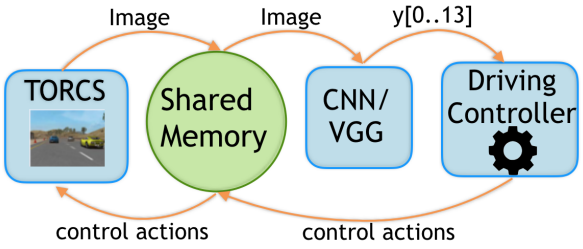


Figure 12. System Architecture for real-time control in TORCS game

Figure 13 is a demo screenshot of running CNN in collaboration with the TORCS game in real-time. A demo video can be found at this playlist `https://www.youtube.com/playlist?list=PL7U9a_VtYNShS4I3ccxeTerufee0KDacE`.

7. Conclusion and Future Work

In this project, we investigated applying transfer learning to direct perception in autonomous driving. We modified VGG architectures and performed thorough experiments on hyper-parameters tuning. We then evaluated our models and achieved comparable MAE result with baseline while outperforming GIST. Finally, we visualized the test results and recorded a video of our VGG model driving host car in TORCS in real-time.

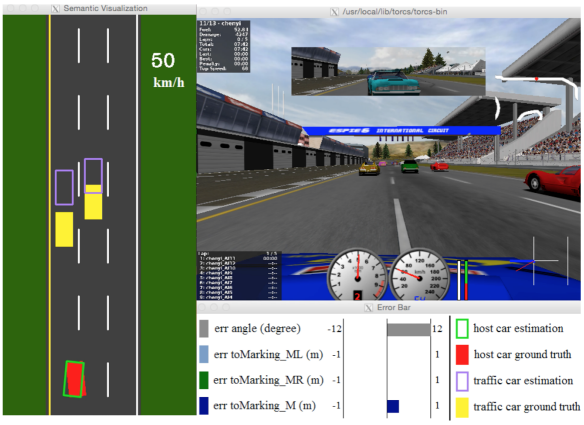


Figure 13. Demo screenshot of running CNN in collaboration with the TORCS game in real-time. Left: visualization, Upper right: game scene, Lower right: error bar and legends.

There are a number of directions we desire to investigate more in the follow-up.

(1) Back-propagate into early convolutional layers of VGG. Currently, we are freezing early layers to speed up each iteration due to time constraints. We hope to investigate the performance of our models when the models are given enough time to train and back-propagate weight updates across the entire network. This is likely to improve performance as the pre-trained weights are optimized for object classifications where some features are not necessarily transferable to understanding driving scenes.

(2) Due to Caffe version limitation, we could not experiment with adding BatchNorm layers or using Adam update to further optimize our VGG model, but these improvements are definitely worth investigation.

(3) Regarding the direct perception approach, we would like to improve the indicators to encode in more holistic perception of the host car to eliminate failure scenarios described in 6.5. For example, we could add one more onMarking indicator that represents the number of lanes to the left of the host car.

(4) A more ambitious direction is to adopt reinforcement learning techniques in further training the network to achieve better autonomous driving.

References

[1] M. Aly. Real time detection of lane markers in urban streets. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 7–12, June 2008.

[2] C. Chen, A. Seff, A. L. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. *CoRR*, abs/1505.00256, 2015.

[3] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *I. J. Robotic Res.*, 32(11):1231–1237, 2013.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[5] D. Held, J. Levinson, and S. Thrun. A probabilistic framework for car detection in images using context and scale. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1628–1634. IEEE, 2012.

[6] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014.

[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[8] A. Krizhevsky, I. Sutskever, and G. 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.

[9] P. Lenz, J. Ziegler, A. Geiger, and M. Roser. Sparse scene flow segmentation for moving object detection in urban environments. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 926–932. IEEE, 2011.

[10] U. Ozguner, T. Acarman, and K. Redmill. *Autonomous ground vehicles*. Artech House, 2011.

[11] D. Pomerleau. ALVINN: an autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1, [NIPS Conference, Denver, Colorado, USA, 1988]*, pages 305–313, 1988.

[12] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[16] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4:2, 2012.

[17] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. *CoRR*, abs/1312.4659, 2013.

[18] B. Wymann, E. Espi, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner. Torcs, the open racing car simulator, 2013.