Curvy DeepPicar

Patrick McNamee
Department of Electrical Engineering and Computer Science
University of Kansas
Lawrence, KS
p678m854@ku.edu

***Abstract*—Convolutional Neural Networks (CNN) focus for vision based driving. Neural network outputs tend to be very noise. Previous work has shown that using the network outputs as a Bèzier curves leads to smooth performances and outperform other network architectures. This work implements a CNN with Bèzier curve outputs on a physical platform for vision navigation and evaluates the results.**

***Index Terms*—machine learning, autonomous vehicles, embdedded platforms, guidance navigation and control**

I. Introduction

Modern car manufacturers are developing autonomous automobiles for future use in both civilian and military purposes. There are various complex environmental aspects that are required for consideration for autonomous agents to succeed in driving tasks such as sensors for object detection and localization, path planning, and state prediction for object avoidance [1]. These computational aspects traditionally rely on computational complex algorithms that may be too slow to implement on a real-time system. An alternative is to use neural networks which are universal approximators [2] to approximate the algorithms for a loss of functional accuracy but an increase in computational throughput. As such, neural networks have been frequently used for autonomous vehicle.

Usage of neural networks in autonomous ground vehicles has been around for decades with ALVINN being one of the first examples of vision-based navigation [3]. ALVINN used a 32 by 32 video input along with a 8 by 32 range finder input feed into a hidden layer of 29 units before the output layer of 46 units. The wheel steering angle was chosen based on the values of the 43 output units and ALVINN was quite successful at its various driving tasks. Modern implementations of autonomous cars, like the one developed by NVIDIA, are empowered by modern computational performance have since moved from the relatively simple network architecture used in ALVINN and currently use CNN which includes convolutional layers as well as significantly larger layers size [4]. While these larger neural networks have allowed for more complex behavior of the autonomous agent, there are still issues that result from the nature of neural networks. Often the outputs of neural networks are rough in the sense small manipulations on the input layer can produce drastically different outputs and there are no guarantees of output behavior in unexplored environments. Still, CNNs are potential methods to work towards fully autonomous driving and there are several implementation techniques to include them.

A. Background

There are various ways for CNN to be implemented for autonomous vision-based automobile navigation as reference in Figure 1. The most direct method is to have the CNN take images and translate them into control inputs such as the steering angle or throttle as in [5]. While direct, this implementation is limited in that the controller is limited by the input rate of the camera systems as well as the network throughput implementation. It is also susceptible to network output roughness which can lead to high control jerk and unnecessarily fatigue physical components. This implementation of a direct controller is referred to in the rest of this work as image-to-point for translating a camera image into a point in the control dimensional space.

Two alternative implementations that are indirect schemes are navigational implementations and are implemented in DeepRacing [6]. Rather than directly control the car, the CNN outputs desired positions to navigate with a lower level control scheme closing the loop. There are two different ways to represent navigational points, discrete and continuous methods. A discrete method is simply to transform the input into a collection of ordered waypoints for navigation. However, CNN with rough outputs will still produce noisy outputs that may cause unwanted behavior. Additionally as the number of waypoints changes, the network architecture needs to be adjusted.

Rather than represent the waypoints or as discrete points, the waypoints can be represented as a continuous polynomial curves. Bèzier curves $\mathbf{B}$ are useful representation as any polynomial of degree d can be implemented with $d+1$ points $\mathbf{P}_k$ by (1) using a parameter t.

$$\mathbf{B}(t) = \sum_{k=0}^{d} \binom{d}{k} (1-t)^{d-k} t^k \mathbf{P}_k \quad t \in [0,1] \tag{1}$$

This representation is advantageous as it can represent the same time window as the discrete waypoint representation but has an infinite number of points so the number of waypoints generated is not upper bounded. Additionally the waypoints are guaranteed to be smooth since they are on a polynomial curve and experimentally DeepRacing showed that this Bèzier representation was the best performing network for CNN in simulated racing [6] but not any physical platform.

While most work is focused on personal or racing automobiles, there have been implementations of vision-based

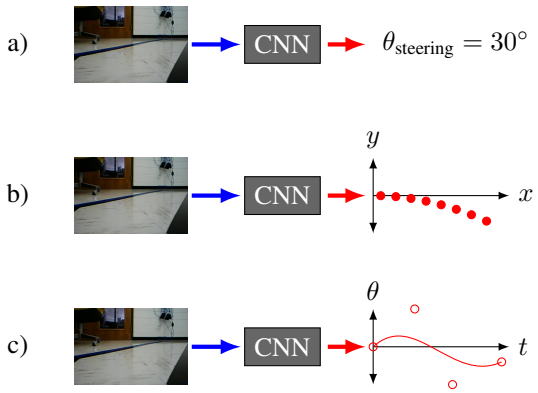

Fig. 1. Various Vision-Based Autonomous Steering Implementations. From top to bottom; a) image to direct control command, b) image to localized waypoints (●), and c) image to curve using poles (○).

navigation on small, radio-controlled cars. These vehicles are significantly cost-effective as research platforms and test embedded systems. Previous work at the University of Kansas has demonstrated implementing the DAVE-2 neural network architecture on a Raspberry Pi for a CNN image-to-point vision navigation system [5]. From a cost standpoint, the DeepPicar platform is ideal for testing a real implementation of a CNN with Bèzier curve output although due to hardware and sensor limitations, an image-to-curve controller will need to used to compare the Bèzier implementation of a controller against a standard image-to-point CNN controller.

B. Motivations

Previous work has shown the use of CNN outputting Bèzier curves for navigation problems in simulated racing outperforms other techniques such as CNN outputting navigation waypoints or direct control networks [6]. However the simulation relies on position estimates which may not be feasible onboard small platforms or in environments with no direct position estimates and the Bèzier curve outputs have, to the knowledge of the author, not been tested on a physical platform currently. Hence this work seeks to implement the a CNN outputting Bèzier curves in a direct control scheme for navigating a track on the DeepPicar.

II. Novel Contributions

This work will implement a CNN outputting Bèzier control curves for an embedded autonomous automobile platform for course navigation which has not been tested before on a physical platform. This new implementation will be tested against a standard CNN with direct control output as used in previous work [5] on the same physical platform.

III. Convolutional Neural Networks

The CNNs are based off of the DAVE-2 CNN architecture from NIVIDIA [4] and is shown in Fig. 2. Previous work has been implemented on a Raspberry Pi for the DeepPicar [5] and this work will use the same implementation as [5] to form a direct comparison with previous implementations. The input to the CNN is a Playstation Eye Camera which generates 320×240 frames which are resized to 200×66 to match the DAVE-2 inputs. Inside the CNN, the image is normalized across the image width, height, and channels before passing though multiple convolution layers. Layers 2, 3, and 4 use 5×5 kernels with a width and height stride of $(2, 2)$ while layers 5 and 6 use 3×3 kernels with a width and height stride of $(1, 1)$. After layer 6, the network is flattened to dense layers whose layer width is reduced until a single output unit. The activation functions used for layers 2 through 10 are rectilinear linear units (ReLU) while the last layer uses a hyperbolic tangent function. These activation functions are consistent with implementations in previous work [4], [5] but the use of ReLU for the hidden layers are consistent with proofs that neural networks are universal approximators [7] and the hyperbolic tangent allows for outputs to be bounded to match the automobiles steering wheel angle limits $[\theta_{\min},\ \theta_{\max}]$ by a linear transformation.

A. Training Datasets

There are two publically datasets available to train radio-control autonomous cars for tracks. The first dataset is a realitively small dataset consisting of only 200 example images but has a continuous steering angle ranging [8] while the other dataset consists of 11,000 example images but only has wheel angles in the set $\{-30°,\ 0°,\ 30°\}$ degrees. As the Bèzier curves deal with continuous outputs, it would be more advantageous from a training perspective to attempt to use Data Augmentation to train the various CNN models on the first dataset and then evaluate the CNN models on the larger second dataset.

¡¡¡Show images from the two data sets¿¿¿
Describe the data augmentation attempt.
¡¡¡Table describing data augmentation accuracy¿¿¿
Describe how handling data in discrete controller.

B. Image-to-Point CNN

TABLE I
Image-to-Point Convolutional Neural Network Architecture

Layer	Layer Type	Dimension	Parameters
1	Normalizer	$66 \times 200 \times 3$	79,200
2	2D Convolution (5×5)	$31 \times 98 \times 24$	1,824
3	2D Convolution (5×5)	$14 \times 47 \times 36$	21,636
4	2D Convolution (5×5)	$5 \times 22 \times 48$	43,248
5	2D Convolution (3×3)	$3 \times 20 \times 64$	43,248
6	2D Convolution (3×3)	$1 \times 18 \times 64$	43,248
7	Flattening	1152	0
8	Dense	100	115,300
9	Dense	50	5,050
10	Dense	10	510
11	Dense	1	11
		Non-trainable	79,200
		Trainable	252,219
		Total	331,419

Brief description of the model architecture and loss function used in training. How data is scaled for this model.

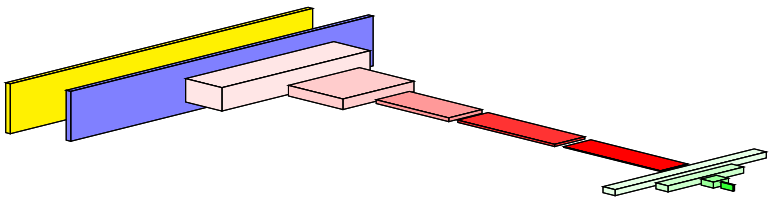

Fig. 2. Visualization of DAVE-2 Neural Network [4]. Yellow is an input image, blue is the regularization layer, red is a convolution layer, and green is a dense layer. Dense layers are exaggerated for viewing and flattening omitted.

TABLE II
BÈZIER CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE

Layer	Layer Type	Dimension	Parameters
1	Normalizer	$66 \times 200 \times 3$	79,200
2	2D Convolution (5×5)	$31 \times 98 \times 24$	1,824
3	2D Convolution (5×5)	$14 \times 47 \times 36$	21,636
4	2D Convolution (5×5)	$5 \times 22 \times 48$	43,248
5	2D Convolution (3×3)	$3 \times 20 \times 64$	43,248
6	2D Convolution (3×3)	$1 \times 18 \times 64$	43,248
7	Flattening	1152	0
8	Dense	100	115,300
9	Dense	50	5,050
10	Dense	4	204
11	Reshape	4×1	0
		Non-trainable	79,200
		Trainable	251,902
		Total	331,102

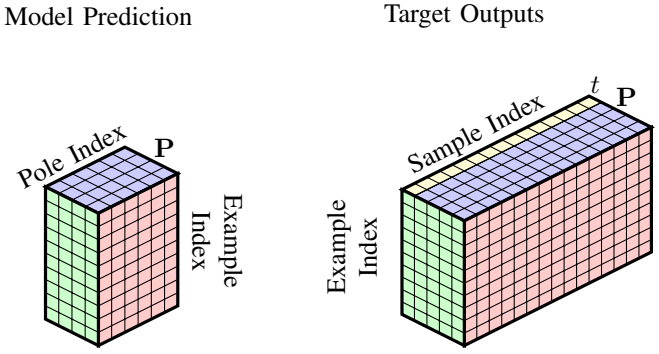

Fig. 3. Implementation of Loss Function

C. Image-to-Curve CNN

Mention how the model is different from a standard image-to-point CNN. How the interval is selected and used for training. How the model can be interpreted

D. Training

Overview of training regiment

IV. EXPERIMENTS

Overview the the quantitative approach for comparison with previous work.

A. Model Accuracy

¡¡¡Video of Results¿¿¿

TABLE III
IMAGE-TO-POINT CNN MODEL (ACC: 20.45%)

		Model Predicted		
		Left	***Center***	***Right***
Record	***Left***	48	31	21
	Center	156	156	188
	Right	54	136	205

TABLE IV
IMAGE-TO-CURVE BÈZIER MODEL (ACC: 38.10%)

		Model Predicted		
		Left	***Center***	***Right***
Record	***Left***	83	46	33
	Center	268	343	313
	Right	177	268	336

B. Physical Driving

Outline of track used. Mention how code was changed. Give some results.

V. RESULTS AND CONCLUSIONS

Mention that the most accurate model was found to be the Bèzier CNN using only the first pole in an image-to-point set up.

Insert

VI. FUTURE WORK

The DeepPicar-v2 dataset is initially limiting; wheel angles are only three variations and there is a one-to-one correspondence between wheel angles and video frames. Curve fitting to noisy data best works with more sampling of points. Rather than constructing the dataset by forming corresponding pairs, have two threads of data collection. The first thread is for the camera at a nominal operational rate for image-to-point usage and the second thread is for

TABLE V
IMAGE-TO-POINT BÈZIER MODEL (ACC: 66.75%)

		Model Predicted		
		Left	***Center***	***Right***
Record	***Left***	60	111	9
	Center	48	667	261
	Right	13	223	608

REFERENCES

[1] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 287–296.

[2] Z. Wang, A. Albarghouthi, G. Prakriya, and S. Jha, "Interval universal approximation for neural networks," 2021.

[3] D. A. Pomerieau, "Alvinn: An autonomous land vehicle in a neural network," Carnegie Mellon University, Tech. Rep., January 1989.

[4] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016.

[5] M. G. Bechtel, E. Mcellhiney, M. Kim, and H. Yun, "Deeppicar: A low-cost deep neural network-based autonomous car," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018, pp. 11–21.

[6] V. S. B. Trent Weiss and M. Behl, "Deepracing ai: Agile trajectory synthesis for autonomous racing," in *International Conference on Intelligent Robotis and Systems (IROS): Workshop on Perception, Learning, and Control for Autonomous Agile Vehicles.* IEEE Robotics and Automation Society, 2020.

[7] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208

[8] D. Tian, "Deeppicar," Online: https://github.com/dctian/DeepPiCar, March 2019.