Lane-Following with DNN

Sergiy Bokhnyak
Informatics
USI
Lugano, Switzerland

Timon Willi
Informatics
USI
Lugano, Switzerland

Abstract—In this project we try to make the thymio drive inside of a lane in an autonomous way using only raw image data from the webcam.

Index Terms—Thymio, CNN, LSTM, Machine Learning, Robotics

I. Introduction

We attempted to use neural networks as controls for the Mighty Thymio [3] with the raw image data as inputs in order to steer the Thymio to drive between two pieces of tape, just like one would drive in a lane on the road. We thought that this would be an interesting problem to see just how easy it is to teach a robot to do something with supervised learning. The idea was to have a clear distinct two line path that the Thymio must follow, and to see to what extent the neural networks would be able to learn the controls and how well what they learned from the training sets, would transfer to actually controlling the robot in real-time when it is actually driving.

A. Data

The track on which we experimented was created by us using dark blue tape on the orange floors of the USI classrooms. We believed that these colors would provide enough contrast for any computer vision algorithm to be able to detect the lines with ease. You can see a picture of the track in Figure 1. The width between the tapes was about 15 cm, so slightly larger than the actual Thymio. The total length of the track is approximately 10 meters, and it was based on designs of some of the simpler Formula 1 tracks.

In order to actually gather the data, we enabled keyboard teleop controls of the Thymio that we used to drive around the track manually. During these runs we recorded the RGB-images that were being published to the ROS node as well as the current angular and linear velocity. The idea was that based on the images, the robot should be able to know whether it should turn, in which direction, and how much. We had two different angular velocities mapped to our controls, one with a weak turn (0.6 angular velocity) and one with a sharper turn (1.0 angular velocity). Our linear velocity during the runs was 0.3. We gathered about 10,000 images, which we then reflected over the y-axis, and switched the sign of angular velocity label–for data augmentation. This way we were guaranteed as many left turns as right turns, and

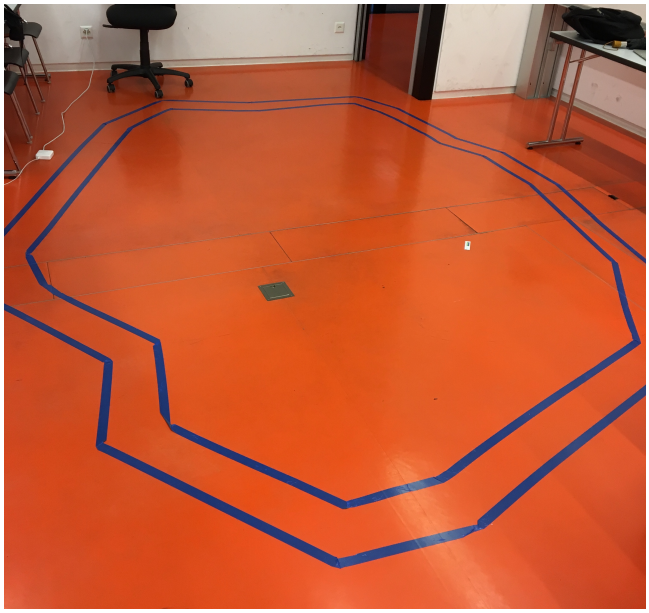

Fig. 1. The track that we used for data collection and testing

in general the more data the better for machine learning methods and especially neural networks. Thus our dataset was about 20,000 labeled images. These pictures we saved as RGB, however we turned them to greyscale for the actual training.

From this we took 2-3 runs and designated them as the test set, which we evaluated the trained models on before running it with the physical robot (our hyperparameter search was done using these test sets, not the actual robot). The errors became quite low though not low enough for us to be very hopeful that it would work in the real world.

II. Models Compared

We trained two different types of neural networks on the same data in order to compare how well each one would be able to learn from the noisy real-world data, and if it would be able to generalize to actual real-time performance. The two models are a purely Convolutional Neural Network and a Convolutional Network with an RNN. The problem is supervised, with the networks attempting to predict the proper angular velocity (as a continuous value) from the raw pixel data.

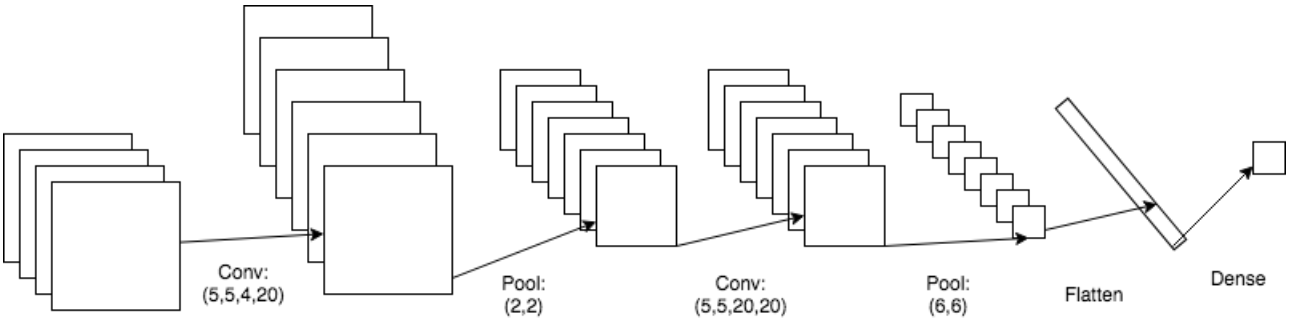

Fig. 2. Convolutional Architecture with *tanh* activation on the output

TABLE I
Dataset Statistics

Directory	# Items	Turn
112657	863	left
113902	3	
114217	27	
114453	80	left
131917	415	left
132532	364	right
132927	2	
133236	600	right
151023	993	left
152356	1123	right
153846	426	left
154420	288	left
154749	1024	left
160122	112	right
160259	1508	right
162403	2188	left
Total Right	3707	
Total Left	6277	

A. CNN

First we considered a simple convolutional network [1]. It had two convolutional layers followed by a dense layer with a *tanh* activation function on the output neuron. The actual specifications can be seen in Figure 2.

The input is a stacked tensor of 4 greyscale images. The reason for this is we thought that a single image may not provide enough contextual information for the CNN to make a correct decision about direction. Since the camera is looking forward with a slight angle down (and not directly down) you could come to a sharp turn and not see the direction in which the road went, however if you have access to the previous 4 frames, you could figure out which way you should turn once you come to the point where the road turns more sharply than the camera can detect without looking straight down.

The reason why we didn't have the camera look straight down was because when looking straight down, if you are directly in the center of the lane you may not see any of the lines but be looking in the middle of them (we noticed this when generating the data). Plus a little bit of perspective on what will come, we thought, might allow the neural network to learn a little bit of planning. Thus we resorted to stacking four frames and passing them to the CNN with a signle label. This label was taken as the average of the four frames' labels. Using the label as the average of the sliding window also ensured smoother controls without jerky direction changes which would have been more likely if we gave just static (categorical) types of labels.

The network overall is rather minimal with only 1720 weights total.

B. RNN+CNN

Next we tried to add an RNN to the mix, in particular the LSTM [2]. We have a similar CNN architecture at the front with only two layers, however there is more aggressive pooling such that the flattened output of the CNN that goes in to the LSTM is much smaller than previously (only 360 neurons). We thought that the LSTM would solve the problem previously mentioned about the field of vision of the robot having to make decisions based on information that it saw moments ago but is not necessarily seeing now. The LSTM has had much success on many sequential tasks so we thought that we would give it a shot here. Because the LSTM has memory in a sense, we did not see a reason to stack many images, however we still wanted it to have some sense of motion so we only stacked two consecutive images, and made the labels the average of the two images labels.

The network overall is also pretty minimal with only 980 trainable parameters.

III. Training and Results on Test Data

Below is a table of our final mean squared errors after training. We trained the CNN for 20,000 steps and the LSTM for 2000 steps (because the LSTM took longer to train since it had to do backpropogation through time).

A mean square error of 0.03 means that the predicted angular velocity is about 0.17 away from the actual value on average (which is pretty large considering that the range of values is between -1 and 1). Looking at the test errors, it would be 0.39 and 0.45 (for the CNN and RNN respectively) average error on each prediction. This is quite

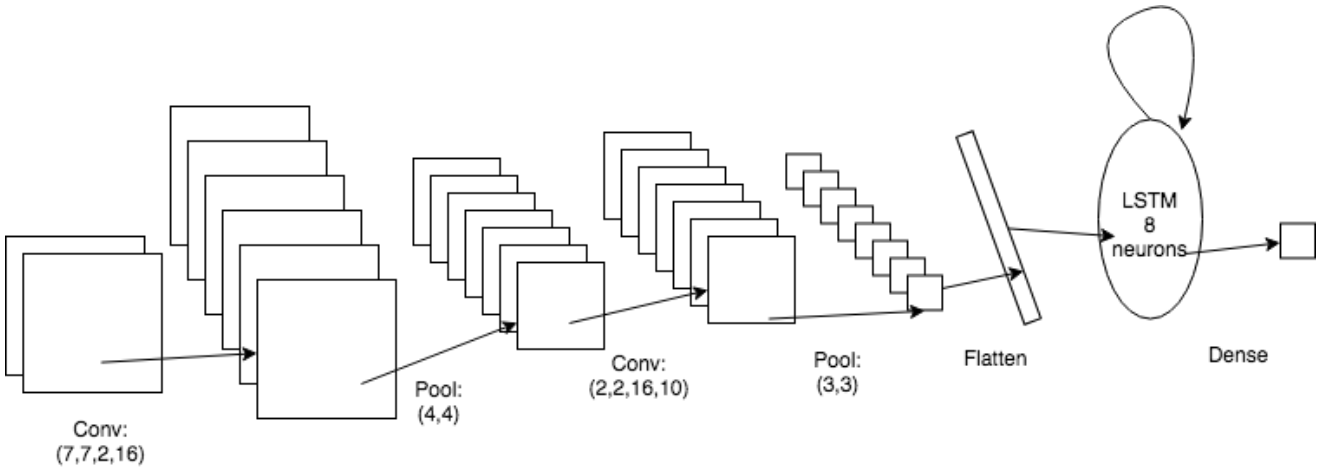

Fig. 3. CNN + RNN with *tanh* activation on the output

TABLE II
Mean Squared Error After training

Model	CNN	RNN
Train Set	0.026	0.03
Test Set	0.15	0.21

large for such a small range for the outputs, thus our hopes for it working in the real world were pretty low already.

IV. Controlling the Thymio in the Real World

When the models were put in control of the Thymio and exposed to our track on which we gathered the data, we found them to not work very well. More specifically they would follow the track for maybe a meter (often times less than that) and then veer off course. The Thymio would however make some turns correctly and sometimes turn corners too early, thereby leaving the lane, however when crossing the road again it would normalize itself, but then would shortly after again go off course.

Overall the results are negative and we conclude that we do not recommend learning driving between two lines using supevised learning and a CNN (with or without an RNN). This may be because our labelling was not perfect and involved a lot of corrections on the fly by the driver, thus maybe the labelling may not have been consistent enough to ensure proper training. Another problem was the fact that though our driving was not perfect during data collection it was still generally bound to the track and thus all of our data assumed that you need to go straight, turn right, or turn left. However in the real world once the controller made an even small mistake and went off course it would be seeing images from outside the track, which is something that it did not see during training, and thus would easily go even more off course because it does not have the means to return to the track. We believe that for the CNN (+RNN) to be able to learn it's way to stay on a track as well as return to the track may be not in a supervised learning scenario but in a reinforcement learning environment, though that would require building a simulation that resembles the track we are training which we did not have time for. In a reinforcement learning framework, the training would involve it going off course and then learning to come back to the track so the controller would be more robust.

Another thing that we realized was that maybe we should have had the CNN classify between 3 classes {straight, right, left} and have a regular Bang-Bang (or figure out a way to use PID) controller actually control the angular velocity of the Thymio to have some separation of the network from the actual robot in order to stabilize its performance, and make the neural networks job a bit easier. Or of course use edge detection and Hough Line detection to parametrize the controller however that's not what we wanted to learn about. What we wanted to see was if it could learn the controls itself by observing a human driver, and our limited experiments say that it cannot (under the simple specifications that we used).

It is important to note that the plain CNN model performed noticeably better in the real world than the RNN model, and it is clear that both models did learn some edge detection and line following, however it was not stable enough to be able to follow these lines consistently.

References

References

[1] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

[2] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[3] Guzzi, Jérôme and Giusti, Alessandro and Di Caro, Gianni A. and Gambardella, Luca M. "Mighty Thymio for Higher-Level Robotics Education" AAAI (2018).