Behavioral Cloning

Behavioral Cloning Project

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- run1.mp4 Video of 2 laps around track 1
- writeup.md summarizing the results

2. Submission includes functional code

When running the Udacity provided simulator in autonomous mode and my drive.py file, the car can be driven autonomously around the track by executing

```
python drive.py model.h5
```

3. Submission code is usable and readable

The model.py file contains the code for loading the necessary data points, as well as training and saving the neural netowrk model for the steering agent. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

The model consists of 5 convolution neural network with 3x3 and 5x5 filter sizes and depths between 24 and 64 (model.py lines 143-148).

Each CNN includes a RELU activation layer to introduce nonlinearity. Before the CNN, the data is normalized in the model using a Keras lambda layer (code line 140), and then passed through a cropping layer (code line 142) to reduce the amount of noise and unnecessary data to be processed.

2. Attempts to reduce overfitting in the model

The model contains dropout layers to reduce overfitting (model.py lines 147, 151, and 153).

The model was trained and validated using the data provided by Udacity, as well as with own data to ensure that the model was not overfitting. This data was collected using the simulator provided by Udacity, using both tracks provided to ensure generalization. The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

The model uses an adam optimizer, so the learning rate was not tuned manually (model.py line 162).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. Around 4 laps of data were collected on trying to satay as close to the center of the road as posible. The track was runned clockwise and counterclockwise, with one lap running at maximum speed (30 MPH) and other at minimum speed (10 MPH).

"Recovery" data was also collected, illustrating to the model the necessity of moving back to the center of the track if the car strays too far to the side.

Model Architecture and Training Strategy

1. Solution Design Approach

As recommended by the videos, and the PDF file provided, the convolution neural network model published by Nvidia was used. This CNN was developed to train the steering agent from a forward-facing camera therefore, this model fits the problem to solve exactly, so it is a good choice for this project.

To combat the possibility of overfitting, the model was modified by adding Keras Dropout layers betwee the last two convolutional layers and between the first three layers of the fully connected network.

Once the model was recreated with the modifications, the first attempt to train it was made. It was noticed that the network was not converging during training. This was a result of a bug in the training set generator that was paring images and angles incorrectly. Also, some of the collected images where not correctly copied and therefore an empty array was passed to the CNN.

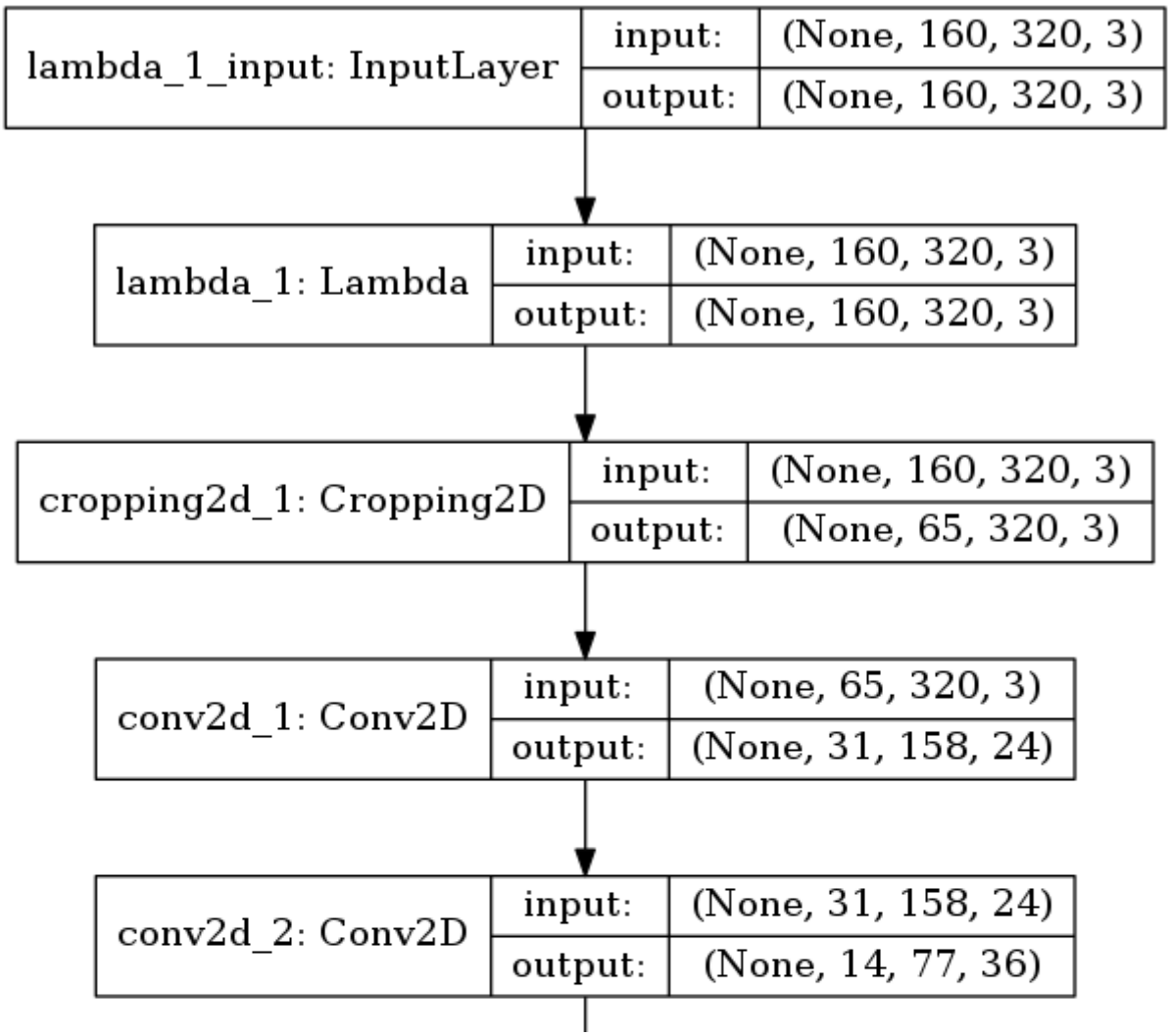
Once the bug was corrected, the model increased in accuracy on both the training and validation sets provided. After two training sessions the model was able to reliably drive the car around the first track.

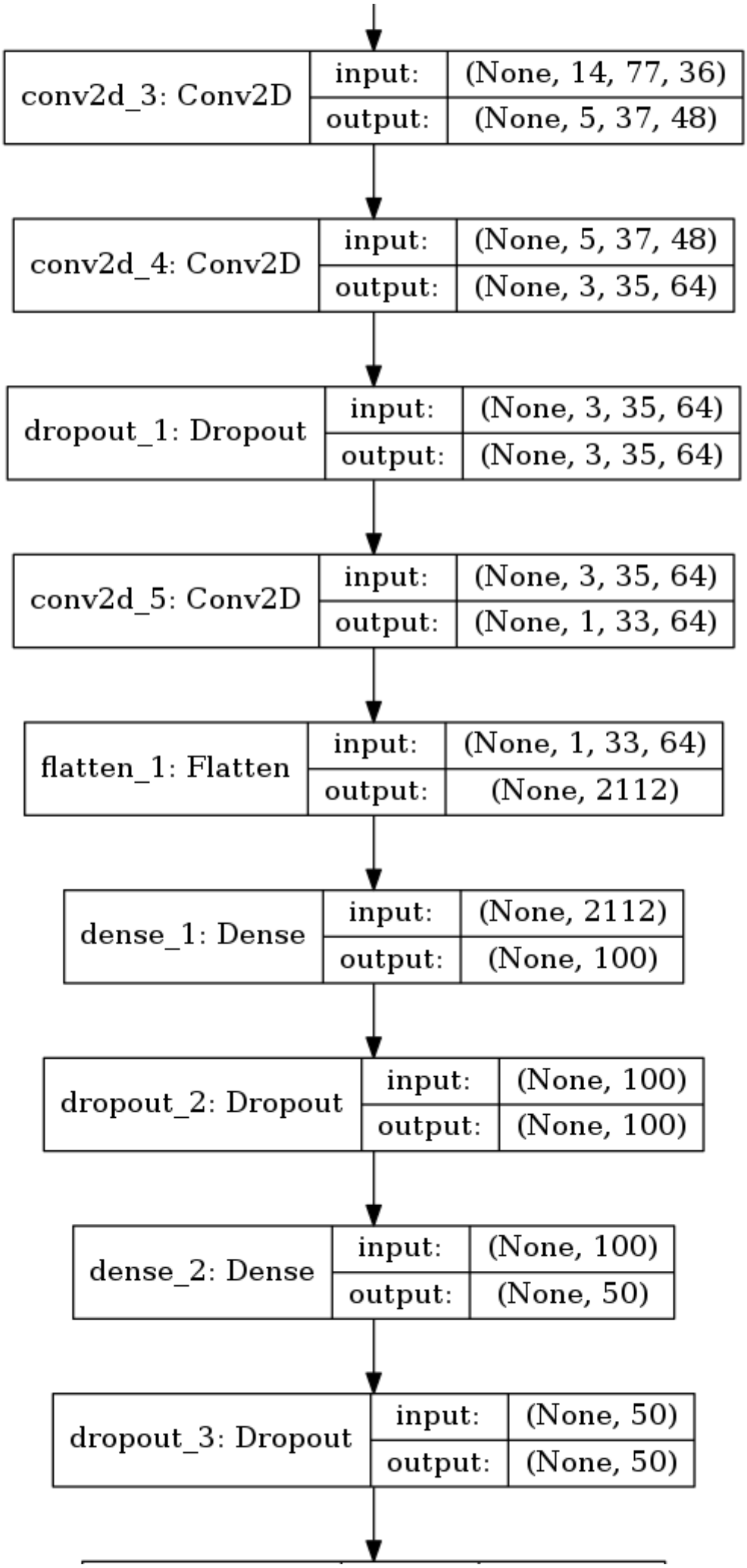
2. Final Model Architecture

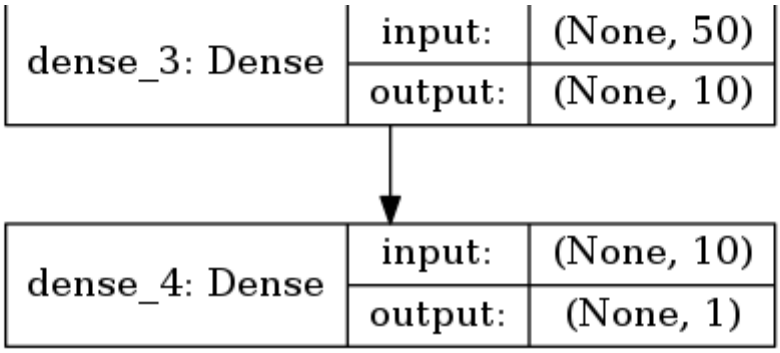
The final model architecture (model.py lines 140-155) consisted of a convolution neural network with the following layers and layer sizes:

```
model = Sequential()
model.add(Lambda(lambda x: (x / 255.0) - 0.5, input_shape=(160, 320, 3)))
model.add(Cropping2D(cropping=((70, 25), (0, 0))))
model.add(Conv2D(24, 5, 5, subsample=(2, 2), activation='relu'))
model.add(Conv2D(36, 5, 5, subsample=(2, 2), activation='relu'))
model.add(Conv2D(48, 5, 5, subsample=(2, 2), activation='relu'))
model.add(Conv2D(64, 3, 3, activation='relu'))
model.add(Dropout(0.4))
model.add(Conv2D(64, 3, 3, activation='relu'))
model.add(Flatten())
model.add(Dense(100))
model.add(Dropout(0.2))
model.add(Dense(50))
model.add(Dropout(0.1))
model.add(Dense(10))
model.add(Dense(1))
```

Here is a visualization of the architecture:




conv2d_3: Conv2D
input: (None, 14, 77, 36)
output: (None, 5, 37, 48)
conv2d_4: Conv2D
input: (None, 5, 37, 48)
output: (None, 3, 35, 64)
dropout_1: Dropout
input: (None, 3, 35, 64)
output: (None, 3, 35, 64)
conv2d_5: Conv2D
input: (None, 3, 35, 64)
output: (None, 1, 33, 64)
flatten_1: Flatten
input: (None, 1, 33, 64)
output: (None, 2112)
dense_1: Dense
input: (None, 2112)
output: (None, 100)
dropout_2: Dropout
input: (None, 100)
output: (None, 100)
dense_2: Dense
input: (None, 100)
output: (None, 50)
dropout_3: Dropout
input: (None, 50)
output: (None, 50)

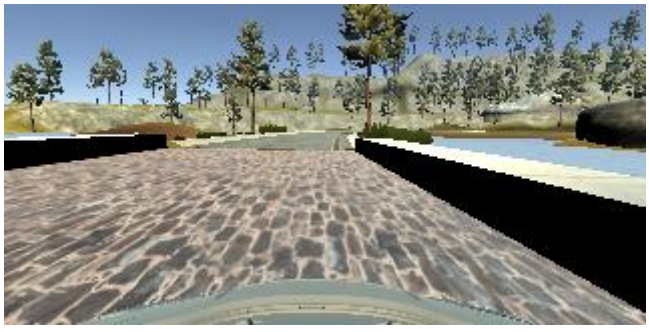


3. Creation of the Training Set & Training Process

To capture good driving behavior, a few laps on track one using center lane driving where recorded. Here is an example image of center lane driving on track one:

Then "recovering" images where recorded placing the car on the edges of the road and driving back to the center for different track lcoations.


Also "recovering images where recorded from parts of the track that could present problems like the bridge and the parts with dirt on the side.

To augment the data set, all the images with an angle different form Zero where flipped.

After the collection and augmentation process, the of data examples totalled ~99,000 examples that where split 80% for training and 20% for validation. This set included adjusted examples from the center camera, both the left and right camera, as well as the flipped versions of the images.