Project: Behavioral Cloning

Patrick Cleeve

Objective

The objective of the project is to develop a neural network to clone human driving behavior on a closed track. The model predicts the required steering angle based on camera images mounted on the front of the vehicle.

The model was trained on a single AWS g2.2xlarge with the udacity-carnd AMI.

The project code is available here:
https://github.com/patrickcleeve/CarND-Term1-BehavioralCloning-P3

Training Data

To collect training data, a human drives the car around the track whilst recording. The training data consists of images from three cameras (left, centre, and right) and the corresponding steering angle. Other measurements collected were not used during the training process.

Example images from the training set (left, centre, right)

The sample training data provided by Udacity has been used as a base for the development and training process (the data is available here). During testing and development, where the autonomous vehicle performed poorly on specific sections (running off the track, crashing or driving erratically), additional training data was collected for the specific areas of the track.

Training Pipeline

Data Collection

THe base data used for training was provided by Udacity, it contained a total of 24,108 images and 8036 measurements. Additional data was collected during the training process, specifically for the turns the model did not perform well on. The additional data collection was limited to less than 500 images for reasons discussed later.

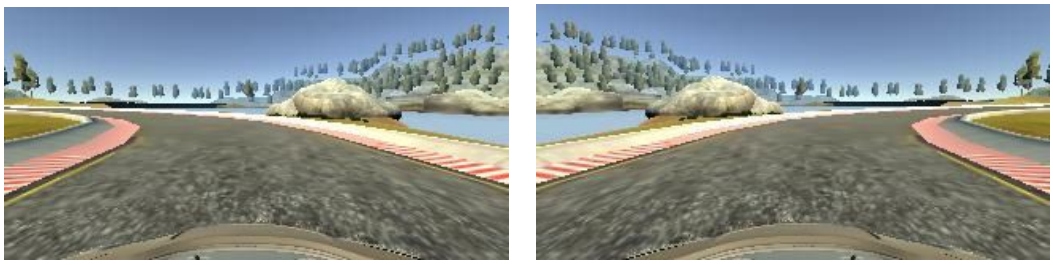
Data Preprocessing

Several image augmentation and preprocessing techniques were evaluated:

Image Normalisation:
The original pixel values of the images range between 0-255 for each RGB channel. These pixel values have been normalised (set between 0 - 1) and mean centred (minus 0.5). This technique helps improve model training convergence. This normalisation is conducted using Keras Lambda layer as shown in Cell 6, Row 6.

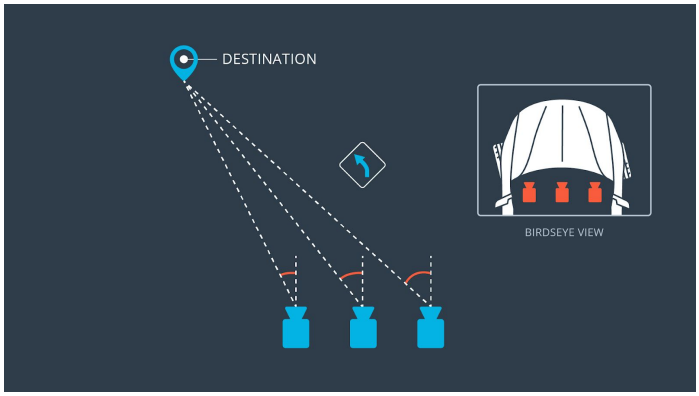
Flipping Images:
The testing track consists primarily of left hand turns, as a result driving around the track counterclockwise makes the model is biased to make left hand turns (and constantly pulls too

hard to the left even when it shouldn't). In addition to driving around the track in the opposite direction, the training data has been augmented by flipping each image horizontally. This technique helps reduce the left hand bias and also doubles the size of the dataset. This augmentation is done using the cv2.flip function, and taking the negative of the corresponding measurement value as shown in Cell 4, Rows 9-10.



Example of flipped image augmentation

Multiple Camera Images:
The car used in the simulator has three cameras mounted on the front of the vehicle (left, centre, and right positions). Apart from increasing the volume of training data, the multiple cameras can improve recovery driving without the need for weaving in and out (which is not a practical method of teaching a car in the real world). The corresponding steering measurement for each camera must be adjusted to account for the offset positions of each of the cameras (see diagram below, source Udacity). The multiple camera approach was tested during development, but not included as part of the final training pipeline.



Example of multi-camera steering angle adjustment (Source)

Image Cropping:
The images collected from the car cameras contain a lot of unnecessary information that it not useful for predicting steering angle for the car. The bottom proportion of the image contains mostly the hood of the vehicle, and the top proportion of the image captures the trees, sky and hills in the background. As these sections of the image are not required for steering angle prediction, the training images have been cropped to improve model performance. The top 70 pixels, and bottom 25 pixels have been cropped from each image using the Keras Cropping2D layer as shown in Cell 6, Row 9.

Example of image cropping

Model Architecture

The final model architecture was:

Keras Layer	**Description**
Input Normalisation and mean centre	160x320x3 RGB Image
Cropping	Crop top 70px, bottom 25px Output: 65x320x3
Convolution (24x5x5) RELU	2x2 strides, Valid padding, Dropout
Convolution (36x5x5) RELU	2x2 strides, Valid padding, Dropout
Convolution (48x5x5) RELU	2x2 strides, Valid padding
Convolution (64x3x3) RELU	Valid padding
Convolution (64x3x3) RELU	Valid padding
Flatten	Output: 1164x1
Dense	Output: 100x1
Dense	Output: 50x1
Dense	Output: 10x1
Output	Output: 1

The model is based off the [NVIDIA End-to-End Learning for Self-Driving Cars].

Model Training Parameters:

Loss Function: Mean Squared Error
Optimiser: Adam
Learning rate: N/A
Epochs: 5

Training

The model was trained on a single AWS g2.2xlarge with the udacity-carnd AMI, running the carnd-term1 Anaconda environment. Initial exploration and training was done on a MacBook Air running the same environment. Training on the AWS GPU took approximately 30 seconds per epoch, in comparison to 30 minutes on the personal device.

Although the GPU provided more efficient training time, a number of trade-offs in utilising the AWS instances were considered. These include:

- **Limited credits available for AWS instances:** Whilst Udacity/AWS generously provided $100 credit for AWS, it was important to balance where the extra processing of the GPU would provide a significant benefit over the laptop training as to not use all the credits. It was important to understand the problem, training challenges and objectives before expending valuable compute resources.

- **Data transfer limits and speed:** As both the collection of training data and evaluation of the model (running in autonomous mode) have to occur on the laptop, training data (images and steering measurements) have to be uploaded to AWS for training, and the model has to be downloaded for evaluation. It was necessary to balance the speed and data size of uploading extra training data collection compared with image augmentation techniques.

The approach taken is discussed in detail in the following section.

Evaluation

The data was split into a training and validation set (80% training, 20% validation). The model was trained for a specific number of epochs. The number of epochs was increased while the validation loss continued to decrease. Model overfitting was identified when the validation loss began to increase, and the number of training epochs was decreased.

The magnitude of the validation loss is not a good indicator of model performance, as it is not a measure of the actual goal of the model (to drive autonomously around the track). As discussed in the project Q&A Video, the validation loss magnitude is more influenced by the similarity of the training data, as opposed to actual model performance.

Therefore, to evaluate the model the performance of running the car around the track must be observed. The model was evaluated on its driving performance, including staying on the track and observing where it performs poorly (crashing, going off road, driving erratically).

Model Architecture and Training Process

The AWS resources required for efficient training (GPU instance) of the convolutional network are expensive to run, and credits available are limited. Whilst the GPU is almost required for model training, it is not for the development of the training pipeline. In addition, limitations on upload speed and data limits (Australian internet) it is time consuming to transfer data between the laptop and AWS. Due to these factors the problem was broken into two separate components:

- Develop a training pipeline
- Develop a model to drive autonomously

Developing a training pipeline

Initially, the primary objective was to develop a working training pipeline. This process involved four steps:

- Data Collection
- Data Preparation
- Model Training
- Model Evaluation

This development process was conducted on the laptop to preserve AWS resources. The initial data used for developing the pipeline was provided by Udacity. The first model tested consisted of the input image directly connected to the output layer. Once the simple model was trained, the model was evaluated using drive.py and the run recorded. The model performed terribly, constantly driving in a left hand circle.

However, this simple model was used to develop the training pipeline involving, importing and preparing data, training a model and successfully exporting the model to run drive the car autonomously around the track.

Once the pipeline was functioning well, and progress was being held back by the time taken to train, the model was transferred to the AWS GPU instance for training.

Develop a model to drive autonomously

As the model was now being trained on the AWS instance, it became necessary to transfer (upload) extra training data to instance. Given the limitations on both upload speed, and data limits it was important to evaluate the specific training examples that would provide the most improvement in the model performance. It became non-trivial to collect more examples. As such, experimentation with additional image augmentation, and more powerful convolutional networks such as the Nvidia end to end model were a more practical approach to improving performance as opposed to collecting more data.

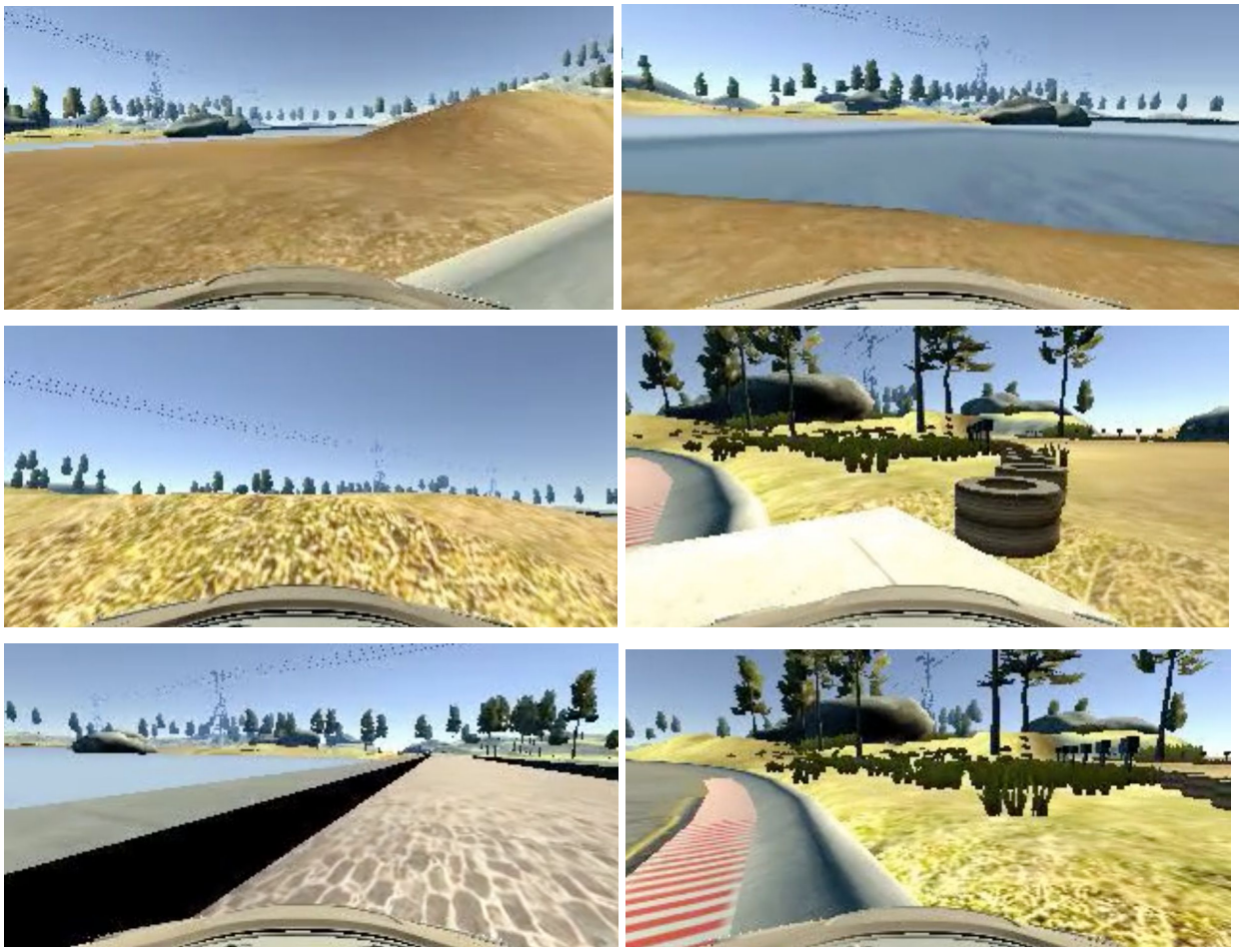
Once the model successful completed the track, additional experimentation with dropout layers on the convolution layers was tested, but ultimately removed from the final model. The model continued to drive successfully around the track until the operation was terminated.

The overall development process is summarised as follows:

Model Architecture	**Preprocessing and Model Parameters**	**Data**	**Loss (Test Outcome)**
Single Fully Connected Layer	Epochs: 5	Base	20+ Drove around in circles
Single Fully Connected Layer	Epochs: 5 Image Normalisation	Base	1.357 Drove off the track
LeNet 5	Epochs: 5 Image Normalisation	Base	0.0111 Drove immediately into the water
LeNet-5 AWS GPU	Epochs: 5 Image Augmentation (Flip) Image Normalisation	Base	0.0121 Got stuck on ledge before the bridge
LeNet-5 AWS GPU	Epochs: 5 Image Augmentation (Flip) Image Normalisation Image Cropping	Base	0.0115 Hit the bridge wall Ran off the track on sharp turn
LeNet-5 AWS GPU	Epochs: 5 Image Augmentation (Flip) Image Normalisation Image Cropping	Base Sharp Turns	0.0205 Ran off the track before sharp turns
Nvidia AWS GPU	Epochs: 4 Image Augmentation (Flip) Image Normalisation Image Cropping	Base	0.0106 Got stuck on sharp turn barrier
Nvidia AWS GPU	Epochs: 5 - 7 Image Augmentation (Flip) Image Normalisation Image Cropping	Base Sharp Turns	0.0103 Ran off the track
Nvidia AWS GPU	Epochs: 5 Image Augmentation (Flip)	Base Sharp Turns	0.0167 Drove successfully around

	Image Normalisation Image Cropping	Last Turn	the track, but close to the edge
Nvidia AWS GPU	Epochs: 5 - 8 Image Augmentation (Flip) Image Normalisation Image Cropping Dropout (Convolutional Layers)	Base Sharp Turns Last Turn	0.113 Ran off the road early in the track
Nvidia AWS GPU	Epochs: 5 Image Augmentation (Flip) Image Normalisation Image Cropping	Base Sharp Turns Last Turn	0.0101 Drove successfully around the track

The following shows some of the failed attempts at the car driving around the track:



Reflection

It was an interesting to train a model with a more open ended objective (i.e. drive around a track), in comparison to a more strict classification task. It forced more thoughtful planning about what it meant for the model to perform well, as opposed to just assessing overall accuracy.

There were some more practical engineering and commercial challenges outside of training the model, such as managing AWS resources and data transfer. It forced you to think about the problem and the project more holistically, and realistically as the resources available were limited.

Future improvements to the performance of the model would include:

- Incorporate multi-camera images into training
- Train and test on more environments (e.g. track 2)
- Use more variables, such as throttle during the training process to better control the car.
- Pre-process the images in different manner, including adjusting brightness

Resubmission

The following changes were made based on reviewer feedback:

- Additional “recovery training data” was collected on sections of the track where the car went near the edges. The model was reloaded and retrained on only these sections of the track.
- Dropout layers (keep_prob=0.5) were added to two of the convolutional layers to reduce the potential of overfitting.
- The pipeline was rewritten to utilise a generator using model training. The model now only loads the images required for a certain batch into memory, allowing usage of larger datasets and using less memory.

Reference

Brownlee, Jason, Adam Optimisation Algorithm
https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Nvidia, End to End Learning for Self Driving Cars
https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

Udacity, Project Repository
https://github.com/udacity/CarND-Behavioral-Cloning-P3

Udacity, Sample Data
https://d17h27t6h515a5.cloudfront.net/topher/2016/December/584f6edd_data/data.zip