Machine Learning Engineer Nanodegree

Capstone Proposal

Michael Virgo
February 13, 2017

Using Deep Learning to Detect Lane Lines

Domain Background

One of my primary areas of focus, and in fact the reason I decided to do the Machine Learning nanodegree originally, is self-driving cars. Although autonomous vehicles have been fantasized about and even attempted almost since the advent of the automobile, they have really began to pick up steam in the past couple of years, with both the incumbent players making moves as well as many technology companies. Self-driving cars use a blend of machine learning, computer vision and various other areas in order to take over the actions of a human driver, providing for safer and more reliable travel. There are many other important benefits, including potential easing of traffic congestion and potential to lower pollution if concepts like ride-sharing continue to increase in use.

As part of the first term of the separate self-driving car nanodegree program, I was tasked with using many different computer vision techniques that require a decent amount of manual input and selection to arrive at the end result (see my Advanced Lane Lines project here). With the knowledge gained from the Machine Learning nanodegree, and some of the Deep Learning course on Udacity's website, I wondered if there might be a better approach to this problem - one directly involving deep learning. Deep learning involves utilizing multiple-layered neural networks, which use mathematical properties to minimize losses from predictions vs. actuals to converge toward a final model, effectively learning as they train on data.

Why this matters

You may say that a fully autonomous vehicle might not necessarily need to directly identify the lane lines - it might otherwise just learn that there are boundaries it is not meant to cross, but not see them as much different from other types of boundaries. If we're skipping straight to a fully autonomous vehicle, this may be true. However, for many consumers, they will likely see more step-by-step changes, and showing them (potentially on an in-vehicle screen) that the car can always sense the lanes will go a long way in getting them comfortable with a computer doing the driving for them. Even short of this, enhanced lane detection could alert an inattentive driver when they drift from their lane.

Problem Statement

Human beings, when fully attentive, do quite well at identifying lane line markings under most driving conditions. Computers are not inherently good at doing the same. However, humans have a disadvantage of not always being attentive (whether it be because of changing the radio, talking to another passenger, being tired, under the influence, etc.), while a computer is. As such, if we can train a computer to get as good as a human at detecting lane lines, since it is already significantly better at paying attention full-time, the computer can take over this job from the human driver. From the computer's perspective, the problem is around detecting a line to fit for both the left and right lane lines.

Datasets and Inputs

The datasets I plan to use will be image frames from video I take from my smartphone. I plan to film in 720p in horizontal/landscape mode, with 720 pixels on the y-axis and 1280 pixels on the x-axis, at 30 fps. In order to cut down on the overall processing needed, I plan to scale down the image sizes, likely down to 25% of the original size. There will definitely be some manual pre-processing involved in order to train my model. As such, I will be calculating the polynomial coefficients for a second-order polynomial for each line (the three outputs per line, times two lines, means each image will have six "labels") in each image. I also plan to use the CV techniques of undistorting and perspective transform to come up with these labels for training images - the hope is that the model will not need these techniques in the fully trained model for new images. Depending on the size of the neural network used, the dataset will likely end up being quite large - at 30 fps, just 5 minutes of video would be 9,000 images, which would be a lot of labelling. I will try a subset of the data first with my labelling technique to ensure the process seems to work before performing the labelling on the full dataset.

I will try to use a mix of both straight lines and various curved lines, as well as various conditions (night vs. day, shadows, rain vs. sunshine) in order to help with the model's overall generalization. I believe these will help to cover more of the real conditions that drivers see every day.

Solution Statement

As I mentioned briefly before, I plan to use deep learning toward the final solution. The reason for this is that a deep learning model may be more effective at determining the important features in a given image than a human being using manual gradient and color thresholds in typical computer vision techniques, as well as other manual programming needed within that process. Specifically, I plan to use a convolutional neural network (CNN), which are very effective at finding patterns within images by using filters to find specific pixel groupings that are important. My aim is for the end result to be both more effective at detecting the lines as well as faster at doing so than common computer vision techniques.

Benchmark Model

I plan to compare the results of the CNN versus the output of the computer vision-based model I used in my SDC nanodegree project (linked to above). I will compare the accuracy/mean-squared error of the lines to each other to see which is more effective, as well as compare the speed of the two techniques (after training, in the case of the CNN). I will also visually compare the output onto the project video from the Advanced Lane Lines project, including attempting to use the CNN on the challenge and harder challenge videos in that project, which my computer vision-based model failed to work on.

Evaluation Metrics

My CNN will be being trained similarly to a regression-type problem, in which it will be given the polynomial coefficients on the training images, and attempt to learn how to extract those values from the given road images. The loss it will minimize is the difference between those actual coefficients and what it predicts (likely using mean-squared error). It will then use only limited computer vision techniques in order to draw the lane back onto the image. As noted above regarding benchmarking against my computer vision-based result, I will also evaluate it directly against that model in both accuracy and speed, as well as on even more challenging videos than my CV-based model was capable of doing.

Project Design

The first step in my project will be data collection. As discussed above, I plan to use my smartphone in a variety of different road settings to gain a good mix of video. Once I have collected enough video, I can then process the video to split out each individual video frame. These frames are the images I will use to train (and validate) my CNN eventually. I plan to scale down the images (starting at 25% of the original size) to reduce processing time, although if I am successful at training a good model with my approach I may attempt a full-scale run.

The second step will be to create labels for the images. There are a thoughts I have here. I plan to calibrate my camera, undistort the image and perspective transform first. I could then draw over the lines in a very distinctive color, and use some basic computer vision techniques (note that this is on training images only - the point is that after training, the CNN will no longer need any computer vision-based components) in order to make a list of pixels making up the lines. This will hopefully be slightly more accurate than relying on various different thresholds in the image as I can make it specific to that single color channel/hue. Once these pixels are detected, I can use numpy.polyfit() to get back the line's coefficients. These coefficients, taken for each line, will be the labels.

In order to check that my labeling is close to a decent fit under this method, I will compare the coefficients achieved through this method to the coefficients returned from my computer vision-based model for the eight test images Udacity included as part of the Advanced Lane

Lines project here. This is a very small portion of data to check against, but will at least allow me to find out very early on if my labelling technique will work as intended, or if I need to take a different approach.

Once I have the data labelled, I will build my convolutional neural network. I plan to begin with a CNN structure similar to that I used in the SDC nanodegree Behavioral Cloning project here, which uses 5 convolutional layers and 4 fully connected layers (along with pooling and dropout within), built with Keras on top of Tensorflow. There are a few changes I already know I need to make, although there will definitely be a decent amount of iteration on the final CNN I use (which luckily Keras makes substantially easier). First, as already discussed, I will need six outputs at the end for my three coefficients for two separate lane lanes, as opposed to just the single output I had for the steering angle in the Behavioral Cloning project. Second, the images from that project were only 360 x 120, so I may need more convolutional layers, larger strides/subsamples, or larger pooling in order to cut down on the number of parameters in the model. Even with 5 convolutional layers, because I used small kernel sizes (the grouping of pixels the CNN considers as a group for a node at each layer), small strides/subsamples, and small pooling, my old CNN model still had nearly 600,000 parameters at the first fully connected layer, which would be vastly increased if the image size is a lot larger.

At this point, after having finished the labelling on a subset of the data, I will test to see whether the CNN actually appears to be learning anything. Although neural networks definitely tend to work better the larger the data size is, if my CNN cannot show at least a little bit of convergence on a subset of the data, I will need to modify my procedures up to this point. Note that this means I will have a training set and a validation set created out of this subset of data using sklearn's train_test_split. If it is working, great, I will move on to using the full dataset.

If the model does appear to be minimizing the error (mean-squared error is my plan), then I will move on to the next step, which is testing the model on completely new data it has never seen before. I will first use the regular project video from the Advanced Lane Lines project and see how the CNN's predictions compare to the computer vision-based model I used before. I will compare both its performance from a visual standpoint (how close it tracks to the lines versus the CV-based model) and from a speed standpoint (how long does it take to process a video vs. the CV-based model). If all goes well, I will also try the CNN's predictions on the challenge video and hard challenge video from the Advanced Lane Lines project, and hope that the end result is a model that can predict faster, and better, than the computer vision-based model, and even become nearly imperceptibly different than a human being would determine it to be.