Finding Lane Lines on the Road

Finding Lane Lines on the Road

The goals / steps of this project are the following:

- Make a pipeline that finds lane lines on the road
- Reflect on your work in a written report

Reflection

1. Describe your pipeline. As part of the description, explain how you modified the draw_lines() function.

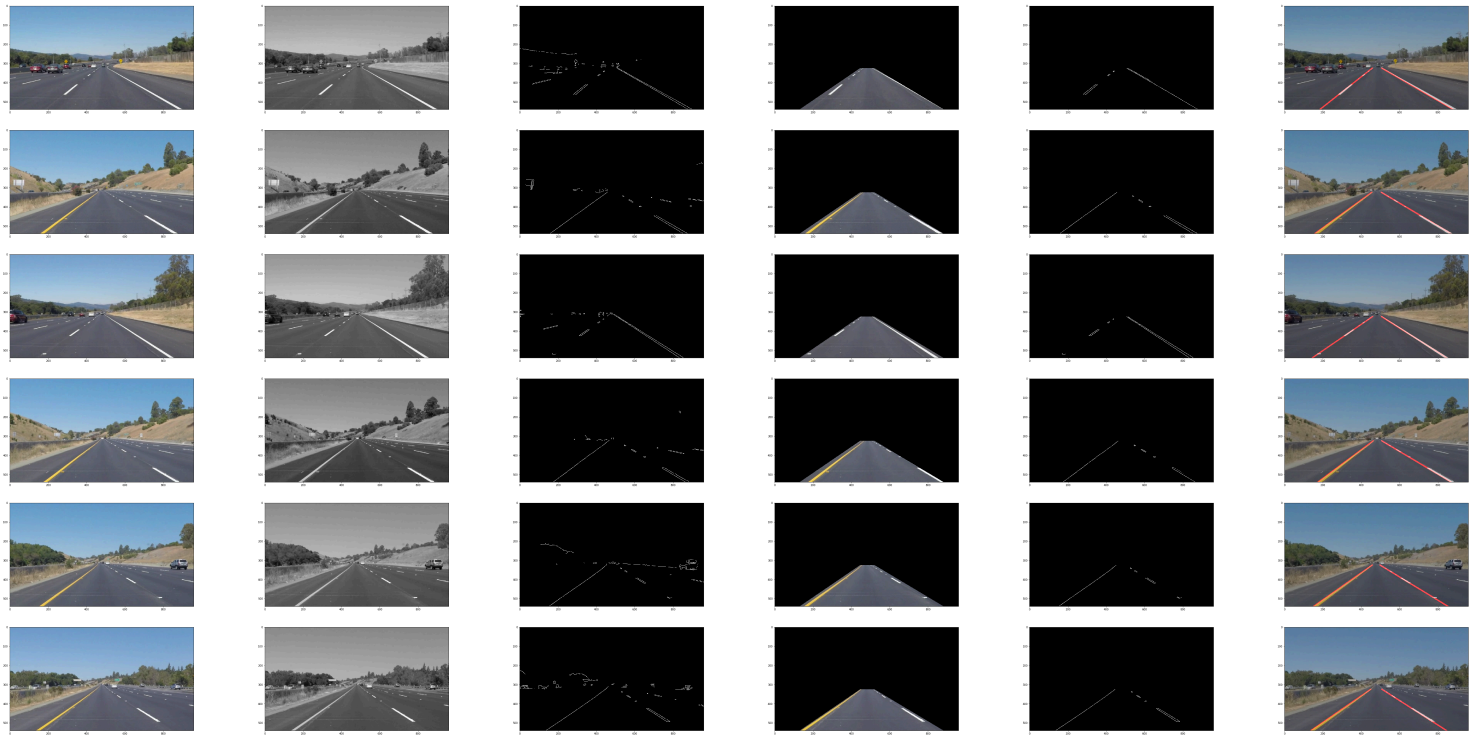
The progression of my pipeline is demonstrated in the figure below. Each row in the figure represents an image from the `test_images` directory. Each column shows processed images after each steps in the pipeline. The first column has the original image. My pipeline consisted of 5 steps:

1. Covert the images to **grayscale** so that it is more convenient for image gradient computation and edge detection. Images after this step are shown in the second column.
2. **[Optional]** Apply **Gaussian blur** to the grayscale images. Gaussian blur is used to reduce image noise and reduce detail. This step is optional because the Canny edge detection function in OpenCV also applies Gaussian blur internally. Images after this step are not shown since it is optional and the images do not look too visually different.
3. Perform **Canny edge detection** on the grayscale images. Images after this step are shown in the third column. The lane lines are correctly identified. However, small edges from the surrounding (e.g., other vehicles and curbside objects) are also picked up by the algorithm.
4. Perform **Region of interest masking** to the detected edges. This step uses a trapezoidal mask to prune away edges from surrounding objects and is critical to the overall success of the pipeline. Images after this step are shown in the fourth and fifth columns. It is evident from the images that by focusing only on the trapezoidal region in front of the car, we are able to remove all the noises from surrounding. The validity of this step requires several assumptions, however:

    - The camera has a fixed focal length and was mounted on a fixed position of the vehicle.
    - The width of road is somewhat fixed.

    - There is no car within the region of interest.

5. Perform **Hough line transform** to detect the lines from edges. Images after this step are shown in the last column. Direct application of the Hough line transform simply marks the dashed lines and does not identify the full extent of the lane and mark them as a single lane line.



In order to **draw a single line** on the left and right lanes, I modified the `draw_lines()` function by:

- **Grouping and filtering**: Separate the lines into two groups based on the sign of their slopes $\frac{y_2-y_1}{x_2-x_1}$. Lines with positive slopes are grouped together and the same for lines with negative slopes. Flat lines (i.e., `abs(slope) < 0.2`) were removed because they are most likely not lane lines.
- **Averaging (linear regression)**: Lines within a group are segments from the same lane line. We recover the original lane line by performing a linear regression using the x and y coordinates of the lines (`numpy.polyfit`). The fitted line $y = kx + b$ is a good representation of the underlying lane line.
- **Extrapolation**: First, we set the lane line's top and bottom end points of y coordinate to be `0.6 * height` and `height` of the image, respectively. Secondly, with these y coordinates we can find their corresponding x coordinates on the fitted line from previous step by $x = \frac{y-b}{k}$.

2. Identify potential shortcomings with your current pipeline

The potential shortcomings of the current pipeline are:

- **Too many hyperparameters** in the pipeline: there are many parameters in every step of the pipeline and

it is time-consuming to tune them.
- **Heuristics** in the pipeline: the region of interest masking is a great idea and a vital step. However, it is based on many assumptions (listed above) and violation of any assumption will lead to failures of the pipeline. For example, for the `challenge.mp4` video, the fixed region of interest included traffic barriers into the region and threw off the line averaging.
- The lane is modeled as a line. This assumption won't work for sharp turns.
- Overall, the tuning is performed on images under similar conditions (e.g., same highway, traffic condition, time of day). So these parameters may not **generalize** to unseen situations. The failure on the challenge video is a manifestation of this generalization issue.

3. Suggest possible improvements to your pipeline

Possible improvements are:

- **End-to-end learning** with deep neural network is a more principled approach comparing to manual tuning. With enough labeled data and minimal preprocessing (i.e., grayscale), we can directly train a model that outputs parameters of lane line (e.g., k and b) given raw images as input. With this approach, we do not need to explicitly design intermediate procedures and abstractions, such as Gaussian blur and Canny edge detection. As a matter of fact, the `convolve` and `pooling` operations in Convolutional Neural Network (CNN) are more powerful procedures than the Gaussian blur and CNN can detect edges automatically, removing the need of tuning parameters.
- **Large and diverse dataset**: the above approach requires a large and diverse labeled dataset so that the model can generalize to all environments and conditions.
- Use second order polynomials to model the lane line.