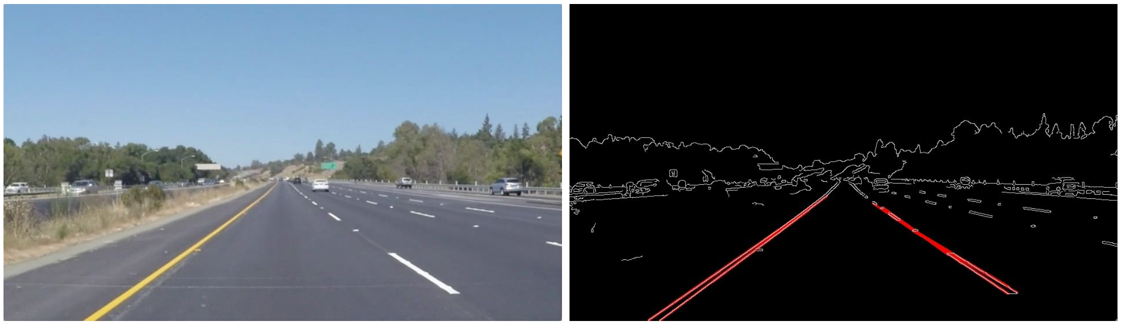
Finding Lane Lines on the Road
Carter Huang, hh548@cornell.edu

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

My pipeline consists of five steps:



- Convert the input image from RGB space to grayscale. Apply Gaussian blurring on the image.
- Utilize Canny Edge Algorithm to find all edges.
- Define a triangular area as region of interest. Initially the region of interest was hard coded. While working on the optional challenge I realize that the region of interest needs to be scaled in proportion to the input image.
- Run Hough Algorithm to find out all lane line candidates.
- Throw away all the candidates that are not inside the region of interest.
- Iterate through all lane line candidates and draw each line on the input image.

I made a few modifications on draw_lines() method:

- Classify lane lines into two buckets: left lane bucket and right lane bucket
- For both buckets, filter out all lane lines with slope $k = \frac{(y_2 - y_1)}{(x_2 - x_1)}, \; k \in (-1, 1)$.
- Computed weighted average for both left and right buckets. The weight is calculated by this equation $w_i = \frac{length_i^2}{\sum_{k \in (0 ,,, N)}^{N} length_k^2}$.
- Draw the weighted average left and right lane lines on the input image.

Identity potential shortcomings with your current pipeline

- Lane lines might not be straight. Our pipeline does not do well matching curve lines.
- Current implementation produces very noisy results when the image is noisy (with trees, signs in the image).
- By using region of interest, we are making an assumption that the lane lines will always be in a predefined area in the image, which is rarely the case in reality.

Suggest potential improvements to your pipeline

- Apply stronger Gaussian blurring to further denoise input image.

- Apply deep learning algorithms (such as mask-RCNN) to come up with the region of interest.
- Apply deep learning algorithms to find lane lines.