Finding Lane Lines on the Road

Reflection

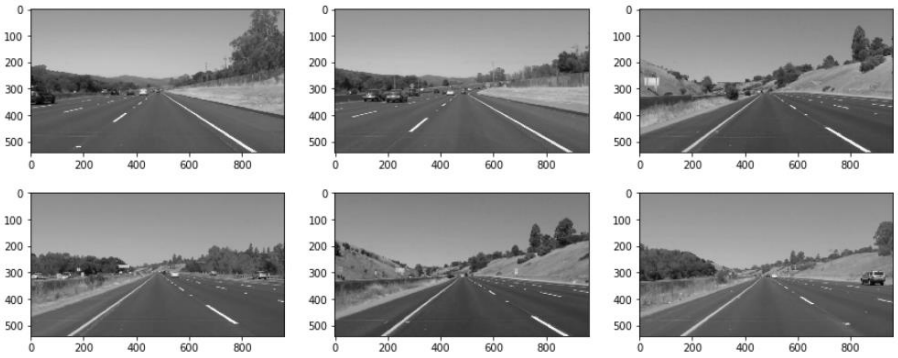
1. Describe your pipeline. As part of the description, explain how you modified the draw_lines() function.

My pipeline consisted of 6 steps.

Step 1:

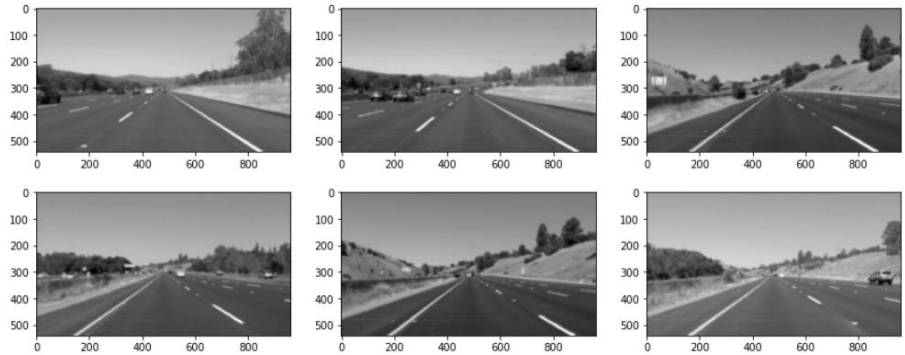
I converted all the test images to grayscale using the helper function.

This step is essential as we need to identify the dark and bright pixels.



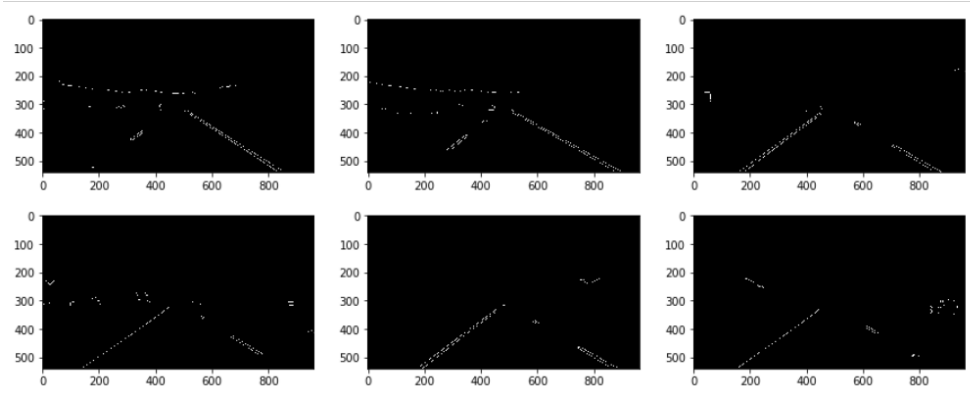
Step 2:

Applied the gaussian blur to the grayscale images to smoothen the image. Since spurious points may cause a problem when drawing the lines.



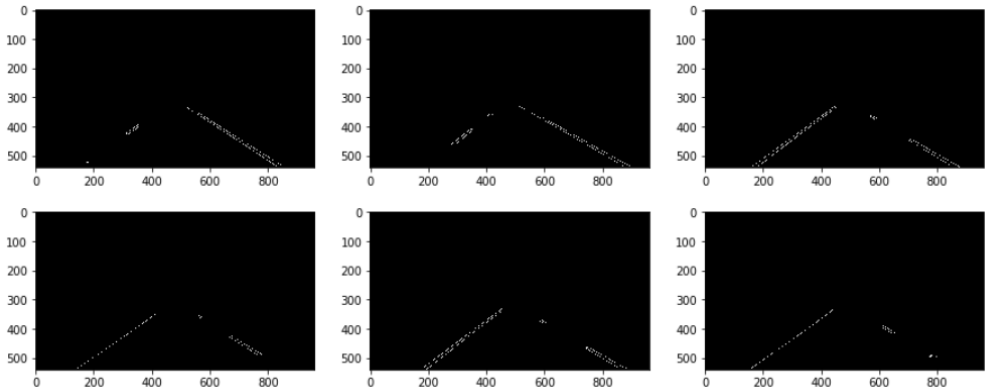
Step 3:

Used the Canny edge detection algorithm to identify the strong edges.



Step 4:

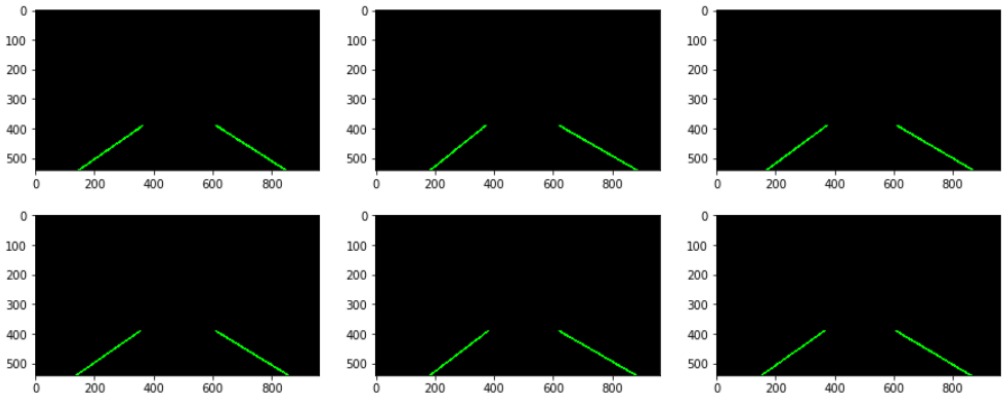
Isolated the region of interest by defining the vertices of the polygon and by using cv2.fillPoly to fill it with pixels.



Step 5:

Used the Hough lines helper function to get the endpoints of the detected lines. Within the function, a blank image was created to draw the lines joining the endpoints of the detected lines.

The draw_lines function was modified in the following ways to get the lanes drawn by joining the endpoints we got from the HoughLinesP function.



Inside Draw Lines:

I wrote 2 additional functions to calculate the lines and the coordinates.

The sequence of steps:

We get the endpoints of the detected line as the output of the HoughLinesP function.

This is a NumPy array.

We pass the image and the array of points as inputs to the calculate_lines function.

Inside calculate_lines:

We have 2 empty arrays to store the coordinates of the left and the right lines

For every detected line we will loop to get the slope and intercepts.

If the slope is <0 we put the values into the left lines empty array and vice versa from the right line array.

After collecting all the slopes and intercepts from the left and right lines we average the values to get only one slope and intercept for each line (i.e.) getting 2 separate line equations.

Then we proceed to calculating coordinates for both the lines.

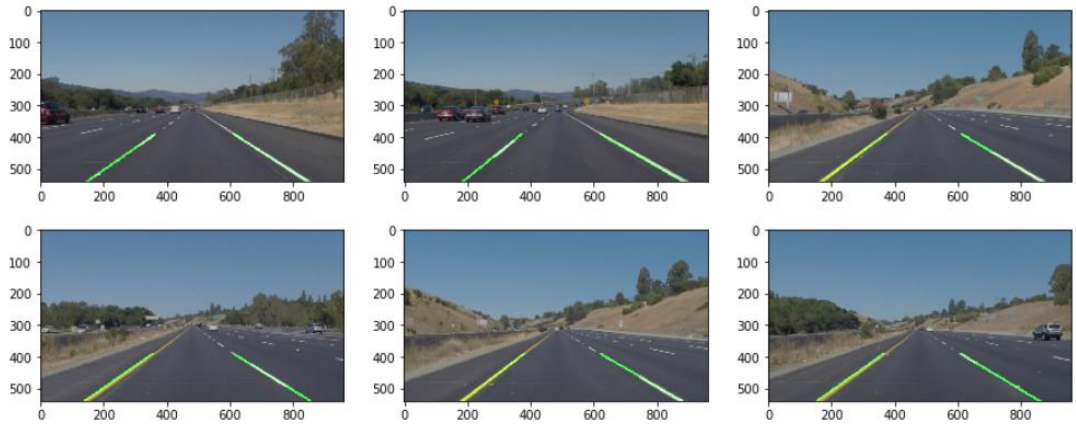
Inside calculate_coordinates:

Based on the slope and intercept values we set the initial and final x and y coordinates of the lines. This will return an array with the starting and ending coordinates to draw the left and right lanes.

Finally, we pass the blank image and the detected endpoints as inputs to the draw_lines function. Using cv2.line we draw the lines by joining the coordinates on the blank image.

Step 6:

Using add Weighted we draw the lines over the lanes of our original image.



On iterating through multiple frames we detect and mark the lanes in the video clips.

2. Identify potential shortcomings with your current pipeline

One potential shortcoming would be what would happen when there are curves in the lanes.

The current pipeline can detect only straight lines.

Another shortcoming could be the setting of the parameters for the Hough Transform. A lot of trial and error attempt was made to arrive at the final values.

Also, since our data is very limited, we do not know how the model would perform when using it to detect lanes under adverse weather conditions.

3. Suggest possible improvements to your pipeline

A possible improvement would be to add some radius calculations or tangent calculations to detect curved lanes.

Another potential improvement could be to have more data to optimize the model using deep learning algorithms. The use of Convolutional Neural Networks and transfer learning will enable us to build more robust models.