Pipeline architecture:

1.Load test images.

2.Apply Color Selection

3.Apply Canny edge detection.

4.Apply gray scaling to the images.

5.Apply Gaussian smoothing.

6.Perform Canny edge detection.

7.Determine the region of interest.

8.Apply Hough transform.

9.Average and extrapolating the lane lines.

10.Apply on video streams.

I'll explain each step in details below.

Environement:

Windows 7

Anaconda 4.3.29

Python 3.6.2

OpenCV 3.1.0

```
import cv2
import numpy as np
from moviepy.editor import VideoFileClip
import os

def list_images(images, cols=2, rows=5, cmap=None):
    """
    Display a list of images in a single figure with matplotlib.
    Parameters:
```

```
    images: List of np.arrays compatible with plt.imshow.
    cols (Default = 2): Number of columns in the figure.
    rows (Default = 5): Number of rows in the figure.
    cmap (Default = None): Used to display gray images.
    """
    plt.figure(figsize=(10, 11))
    for i, image in enumerate(images):
        plt.subplot(rows, cols, i+1)
        # Use gray scale color map if there is only one channel
        cmap = 'gray' if len(image.shape) == 2 else cmap
        plt.imshow(image, cmap=cmap)
        plt.xticks([])
        plt.yticks([])
    plt.tight_layout(pad=0, h_pad=0, w_pad=0)
    plt.show()


def HSL_color_selection(image):
    """
    Apply color selection to the HSL images to blackout everything except for white and yellow lane lines.
    Parameters:
        image: An np.array compatible with plt.imshow.
    """
    # Convert the input image to HSL
    converted_image = convert_hsl(image)


    # White color mask
    lower_threshold = np.uint8([0, 200, 0])
    upper_threshold = np.uint8([255, 255, 255])
    white_mask = cv2.inRange(converted_image, lower_threshold, upper_threshold)
```

```
    # Yellow color mask
    lower_threshold = np.uint8([10, 0, 100])
    upper_threshold = np.uint8([40, 255, 255])
    yellow_mask = cv2.inRange(converted_image, lower_threshold, upper_threshold)

    # Combine white and yellow masks
    mask = cv2.bitwise_or(white_mask, yellow_mask)
    masked_image = cv2.bitwise_and(image, image, mask=mask)

    return masked_image

def region_selection(image):
    """
    Determine and cut the region of interest in the input image.
    Parameters:
        image: An np.array compatible with plt.imshow.
    """
    mask = np.zeros_like(image)
    # Defining a 3 channel or 1 channel color to fill the mask with depending on the input image
    if len(image.shape) > 2:
        channel_count = image.shape[2]
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255
    # We could have used fixed numbers as the vertices of the polygon,
    # but they will not be applicable to images with different dimensions.
    rows, cols = image.shape[:2]
    bottom_left = [cols * 0.1, rows * 0.95]
    top_left = [cols * 0.4, rows * 0.6]
    bottom_right = [cols * 0.9, rows * 0.95]
    top_right = [cols * 0.6, rows * 0.6]
```

```
    vertices = np.array([[bottom_left, top_left, top_right, bottom_right]], dtype=np.int32)
    cv2.fillPoly(mask, vertices, ignore_mask_color)
    masked_image = cv2.bitwise_and(image, mask)
    return masked_image


def hough_transform(image):
    """
    Determine and cut the region of interest in the input image.
    Parameters:
        image: The output of a Canny transform.
    """
    rho = 1             # Distance resolution of the accumulator in pixels.
    theta = np.pi/180    # Angle resolution of the accumulator in radians.
    threshold = 20       # Only lines that are greater than threshold will be returned.
    minLineLength = 20   # Line segments shorter than that are rejected.
    maxLineGap = 300     # Maximum allowed gap between points on the same line to link them
    return cv2.HoughLinesP(image, rho=rho, theta=theta, threshold=threshold,
                           minLineLength=minLineLength, maxLineGap=maxLineGap)


def average_slope_intercept(lines):
    """
    Find the slope and intercept of the left and right lanes of each image.
    Parameters:
        lines: The output lines from Hough Transform.
    """
    left_lines = []  # (slope, intercept)
    left_weights = []  # (length,)
    right_lines = []  # (slope, intercept)
    right_weights = []  # (length,)

    for line in lines:
```

```
        for x1, y1, x2, y2 in line:
            if x1 == x2:
                continue
            slope = (y2 - y1) / (x2 - x1)
            intercept = y1 - (slope * x1)
            length = np.sqrt(((y2 - y1) ** 2) + ((x2 - x1) ** 2))
            if slope < 0:
                left_lines.append((slope, intercept))
                left_weights.append((length))
            else:
                right_lines.append((slope, intercept))
                right_weights.append((length))
    left_lane = np.dot(left_weights, left_lines) / np.sum(left_weights) if len(left_weights) > 0 else
None
    right_lane = np.dot(right_weights, right_lines) / np.sum(right_weights) if len(right_weights) > 0
else None
    return left_lane, right_lane


def pixel_points(y1, y2, line):
    """
    Converts the slope and intercept of each line into pixel points.
    Parameters:
        y1: y-value of the line's starting point.
        y2: y-value of the line's end point.
        line: The slope and intercept of the line.
    """
    if line is None:
        return None
    slope, intercept = line
    x1 = int((y1 - intercept) / slope)
    x2 = int((y2 - intercept) / slope)
    y1 = int(y1)
```

```
    y2 = int(y2)
    return ((x1, y1), (x2, y2))

def lane_lines(image, lines):
    """
    Create full length lines from pixel points.
    Parameters:
        image: The input test image.
        lines: The output lines from Hough Transform.
    """
    left_lane, right_lane = average_slope_intercept(lines)
    y1 = image.shape[0]
    y2 = y1 * 0.6
    left_line = pixel_points(y1, y2, left_lane)
    right_line = pixel_points(y1, y2, right_lane)
    return left_line, right_line

def draw_lane_lines(image, lines, color=[255, 0, 0], thickness=12):
    """
    Draw lines onto the input image.
    Parameters:
        image: The input test image.
        lines: The output lines from Hough Transform.
        color (Default = red): Line color.
        thickness (Default = 12): Line thickness.
    """
    line_image = np.zeros_like(image)
    for line in lines:
        if line is not None:
            cv2.line(line_image, *line, color, thickness)
    return cv2.addWeighted(image, 1.0, line_image, 1.0, 0.0)
```

```
def frame_processor(image):
    """
    Process the input frame to detect lane lines.
    Parameters:
        image: Single video frame.
    """
    color_select = HSL_color_selection(image)
    gray = gray_scale(color_select)
    smooth = gaussian_smoothing(gray)
    edges = canny_detector(smooth)
    region = region_selection(edges)
    hough = hough_transform(region)
    result = draw_lane_lines(image, lane_lines(image, hough))
    return result

def process_video(test_video, output_video):
    """
    Read input video stream and produce a video file with detected lane lines.
    Parameters:
        test_video: Input video.
        output_video: A video file with detected lane lines.
    """
    input_video = VideoFileClip(os.path.join('test_videos', test_video), audio=False)
    processed = input_video.fl_image(frame_processor)
    processed.write_videofile(os.path.join('output_videos', output_video), audio=False)
```