Robust Lane Detection from Continuous Driving Scenes Using Deep Neural Networks

Abstract:

Lane detection in driving scenes is an important module for autonomous vehicles and advanced driver assistance systems. In recent years, many sophisticated lane detection methods have been proposed. However, most methods focus on detecting the lane from one single image, and often lead to unsatisfactory performance in handling some extremely-bad situations such as heavy shadow, severe mark degradation, serious vehicle occlusion, and so on. In fact, lanes are continuous line structures on the road. Consequently, the lane that cannot be accurately detected in one current frame may potentially be inferred out by incorporating information of previous frames. To this end, we investigate lane detection by using multiple frames of a continuous driving scene, and propose a hybrid deep architecture by combining the convolutional neural network (CNN) .Specifically, information of each frame is abstracted by a CNN block, and the CNN features of multiple continuous frames, holding the property of time-series, are then fed into the CNN block for feature learning and lane prediction. Extensive experiments on two large-scale datasets demonstrate that, the proposed method outperforms the competing methods in lane detection, especially in handling difficult situations.

Existing Method:

In recent years, a number of lane-detection methods have been proposed with sophisticated performance as reported in the literatures. Among these methods, some represent the lane structure with geometry models, some formulate lane detection as energy minimization problems, and some segment the lane by using supervised learning strategies, and so on. However, most of these methods limit their solutions by detecting road lanes in one current frame of the driving scene, which would lead to low performance in handling challenging driving scenarios such as heavy shadows, severe road mark degradation, serious vehicle occlusion, as shown in the top three images. In these situations, the lane may be predicted with false direction, or may be detected in partial, or even cannot be detected at all. One main reason is that, the information provided by the current frame is not enough for accurate lane detection or prediction.

Disadvantage:

- Ø Among these methods, some represent the lane structure with geometry models, some formulate lane detection as energy minimization problems, and some segment the lane by using supervised learning strategies, and so on.
- Ø Meanwhile, we notice that deep learning as an emerging technology has demonstrated the state-of-the-art, human-competitive, and sometimes better-than-human performance in solving many

computer vision problems such as object detection image classification/retrieval and semantic segmentation.

Proposed Method:

Based on the discussion above, a hybrid deep neural network is proposed for lane detection by using multiple continuous driving scene images. The proposed hybrid deep neural network combines the DCNN. In a global perspective, the proposed network is a DCNN, which takes multiple frames as an input, and predict the lane of the current frame in a semantic-segmentation manner. A fully convolution DCNN architecture is presented to achieve the segmentation goal. It contains an encoder network and a decoder network, which guarantees that the final output map has the same size as the input image. In a local perspective, features abstracted by the encoder network of DCNN. The output of DCNN is supposed to have fused the information of the continuous input frames, and is fed into the decoder network of the DCNN to help predict the lanes.

Advantage:

- Ø The advantage of this model is that, the predicted lanes are thin and accurate, avoiding marking large soft boundaries as usually brought by CNNs.

- Ø UNet- ConvLSTM outperforms other methods in terms of precision for all scenes with a large margin of improvement, and achieves highest F1 values in most scenes, which indicates the advantage of the proposed models.
- Ø Meanwhile, the experimental results demonstrated the advantages of CNN in sequential feature learning and target-information prediction in the context of lane detection. Compared with other models, the proposed models showed higher performance with relatively higher precision, recall, and accuracy values.

SYSTEM SPECIFICATION:

SOFTWARE REQUIREMENTS:

Operating system :	Windows 7 Ultimate.
Coding Language :	Python.
Front-End :	Python console.

HARDWARE REQUIREMENTS:

System :	Pentium IV 2.4 GHz.
Hard Disk :	40 GB.
Floppy Drive :	1.44 Mb.
Monitor :	14’ Colour Monitor.
Mouse :	Optical Mouse.

Ram : 512 Mb.

Introduction:

Many research groups from both academia
and industry have invested large amount of resources to
develop advanced algorithms for driving scene understanding,
targeting at either an autonomous vehicle or an advanced driver
assistance system (ADAS). Among various research topics of
driving scene understanding, lane detection is a most basic
one. Once lane positions are obtained, the vehicle will know
where to go, and avoid the risk of running into other lanes.
In recent years, a number of lane-detection methods have
been proposed with sophisticated performance as reported in
the literatures. Among these methods, some represent the lane
structure with geometry models [2], [3], some formulate lane
detection as energy minimization problems [4], [5], and some
segment the lane by using supervised learning strategies [6]–
[9], and so on. However, most of these methods limit their
solutions by detecting road lanes in one current frame of

the driving scene, which would lead to low performance in handling challenging driving scenarios such as heavy shadows, severe road mark degradation, serious vehicle occlusion, as shown in the top three images in Fig. 1. In these situations, the lane may be predicted with false direction, or may be detected in partial, or even cannot be detected at all. One main reason is that, the information provided by the current frame is not enough for accurate lane detection or prediction.

Generally, road lanes are line structures and continuous on the pavement, appeared either in solid or dash. As the driving scenes are continuous and are largely overlapped between two neighboring frames, the position of lanes in the neighboring frames are highly related. More precisely, the lane in the current frame can be predicted by using multiple previous frames, even though the lane may suffer from damage or degradation brought by shadows, stains and occlusion. This motivates us to study lane detection by using images of a continuous driving scene. A hybrid deep neural net work is proposed for lane detection by using multiple contin

uous driving scene images. The proposed hybrid deep neural

network combines the DCNN and the DRNN. In a global

perspective, the proposed network is a DCNN, which takes

multiple frames as an input, and predict the lane of the current

frame in a semantic-segmentation manner. A fully convolution

DCNN architecture is presented to achieve the segmentation

goal. It contains an encoder network and a decoder network,

which guarantees that the fifinal output map has the same size

as the input image. In a local perspective, features abstracted

by the encoder network of DCNN are further processed by

a DRNN. A long short-term memory (LSTM) network is

employed to handle the time-series of encoded features. The

output of DRNN is supposed to have fused the information

of the continuous input frames, and is fed into the decoder

network of the DCNN to help predict the lanes.

The main contributions of this paper lie in three-fold:

• First, for the problem that lane cannot be accurately de tected using one single image in the situation of shadow,

road mask degradation and vehicle occlusion, a novel

method that using continuous driving scene images for lane

detection is proposed. As more information can be extracted

from multiple continuous images than from one current

image, the proposed method can more accurately predicted

the lane comparing to the single-image-based methods,

especially in handling the above mentioned challenging

situations.

• Second, for seamlessly integrating the DRNN with DCNN,

a novel fusion strategy is presented, where the DCNN

consists of an encoder and a decoder with fully-convolution

layers, and the DRNN is implemented as an LSTM

network. The DCNN abstracts each frame into a low

dimension feature map, and the LSTM takes each feature map as a full-connection layer in the time line and

recursively predicts the lane. The LSTM is found to be

very effective for information prediction, and signifificantly improve the performance of lane detection in the semantic

segmentation framework.

• Third, two new datasets are collected for performance

evaluation. One dataset collected on 12 situations with

each situation containing hundreds of samples. The other

dataset are collected on rural roads, containing thousands of

samples. These datasets can be used for quantitative eval

uation for different lane-detection methods, which would

help promote the research and development of autonomous

driving. Meanwhile, another dataset, the TuSimple dataset,

is also expanded by labeling more frames.

Literature Survey:

A Learning-Based Approach for Lane Departure Warning Systems With a Personalized Driver Model

We propose a learning-based approach to predicting unintended lane-departure behaviors (LDB) and the chance for drivers to bring the vehicle back to the lane. First, in this approach, a personalized driver model for lane-departure and lane-keeping behavior is established by combining the Gaussian mixture model and the hidden Markov model. Second, based on this model, we develop an online model-based prediction algorithm to predict the forthcoming vehicle trajectory and judge whether the driver will demonstrate an LDB or a DCB. We also develop a warning strategy based on the model-based prediction algorithm that allows the lane-departure warning system to be acceptable for drivers according to the predicted trajectory.

Fast learning method for convolutional neural networks using extreme learning machine and its application to lane detection

we suggest a new learning algorithm for CNNs using an extreme learning machine (ELM). The ELM is a fast learning method used to calculate network weights between output and hidden layers in a single iteration and thus, can dramatically reduce learning time while producing accurate results with minimal training data. A conventional ELM can be applied to networks with a single hidden layer; as such, we propose a stacked ELM architecture in the CNN framework. Further, we modify the backpropagation algorithm to find the targets of hidden layers and effectively learn network weights while maintaining performance. Experimental results confirm that the proposed method is effective in reducing learning time and improving performance.

You Only Look Once: Unified, Real-Time Object Detection

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background.

INPUT AND OUTPUT DESIGN INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

• What data should be given as input?

• How the data should be arranged or coded?

• The dialog to guide the operating personnel in providing input.

• Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES

1.Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

3.When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus, the objective of input design is to create an input layout that is easy to follow.

OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making. 1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements. 2.Select methods for presenting information. 3.Create document, report, or other formats that contain information produced by the system. The output form of an information system should accomplish one or more of the following objectives.

• Convey information about past activities, current status or projections of the

• Future.

• Signal important events, opportunities, problems, or warnings.

• Trigger an action.

• Confirm an action.

SOFTWARE REQUREIMENTS ANALYSIS

Problem Specification Nowadays as IT industries are setting a new peek in the market by bringing new technologies and products in the market. In this study, we will be giving a guidance to the people who are unable to choose the carrier on their own. In this we will be giving the input as the knowledge in particular subject and we will receive the output as the carrier related to the knowledge the particular person had such as developer, designer, manager etc... we can also provide a single input in which we are good at that will be giving the output related to it for example if a person had only the programming skills and he does not know anything related to other technologies he can become a good developer.

Modules and Their Functionalities

1. Data Analysis

After collecting datasets from various resources. Dataset must be pre-processing before training to the model. The data pre-processing can be done by various stages, begins with reading the collected dataset the process continues to data cleaning. In data cleaning the datasets contain some redundant attributes, those attributes are not considering for career prediction. So, we have to drop unwanted attributes and datasets containing some missing values we need to drop these missing values or fill with unwanted nan values in order to get better accuracy.

2. Data Preprocessing

After collecting datasets from various resources. Dataset must be pre-processing before training to the model. The data pre-processing can be done by various stages, begins with reading the collected dataset the process continues to data cleaning. In data cleaning the datasets contain some redundant attributes, those attributes are not considering for career prediction. So, we have to drop unwanted attributes and datasets containing some missing Values we need to drop these missing values or fill with unwanted nan values in order to get better accuracy.

3. Machine Learning Algorithm for Prediction

Machine learning predictive algorithms has highly optimized estimation has to be likely outcome based on trained data. Predictive analytics is the use of data, statistical algorithms and machine learning techniques to identify the likelihood of future outcomes based on historical data. The goal is to go beyond knowing what has happened to providing a best assessment of 9 what will happen in the future. In our system we used supervised machine learning algorithm having subcategories as classification and regression

Functional Requirements

It provides the users a clear statement of the functions required for the system in order to solve the project information problem it contains a complete set of requirements for the applications. A requirement is condition that the application must meet for the customer to find the application satisfactory. A requirement has the following characteristics

1. It provides a benefit to the origination.

2. It describes the capabilities the application must provide in business terms.

3. It does not describe how the application provides that capability.

4. It is stated in unambiguous words. Its meaning is clear and understandable.

5. It is verifiable.

Non-Functional Requirements

Career recommendation non-functional requirements, like interests he has, how hours he can work likewise, with today’s IT projects, to determine non-functional requirements, like availability, the approach requires that the designer 1 st determine the scope: does the whole solution or only part of it need to be architected to meet minimum levels?

1. This is done through 4 steps:

2. Identify the critical areas of solutions

3. Identify the critical components within each critical area.

4. Determine each component's availability and risk.

5. Model worst-case failure scenarios.

Feasibility Study

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

1. ECONOMICAL FEASIBILITY

2. TECHNICAL FEASIBILITY

3. SOCIAL FEASIBILITY

1. Economic Feasibility
This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

2. Technical Feasibility
This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

3. Social Feasibility
The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

Modules:

Advanced driver assistance system:

WITH the rapid development of high-precision optic sensors and electronic sensors, high-efficient and high effective computer vision and machine learning algorithms, real-time driving scene understanding has become more and more realistic to us. Many research groups from both academia and industry have invested large amount of resources to develop advanced algorithms for driving scene understanding, targeting at either an autonomous vehicle or an advanced driver assistance system (ADAS). Among various research topics of driving scene understanding, lane detection is a most basic one. Once lane positions are obtained, the vehicle will know where to go, and avoid the risk of running into other lanes.

Lane detection:

In recent years, a number of lane-detection methods have been proposed with sophisticated performance as reported in the literatures. Among these methods, some represent the lane structure with geometry models, some formulate lane detection as energy minimization problems, and some segment the lane by using supervised learning strategies, and so on. However, most of these methods limit their solutions by detecting road lanes in one current frame of the driving scene, which would lead to low performance in handling challenging driving scenarios such as heavy shadows, severe road mark degradation, serious vehicle occlusion, as shown in the top three images. In these situations, the lane may be predicted with false direction, or may be detected in partial, or even cannot be detected at all. One main reason is that, the information provided by the current frame is not enough for accurate lane detection or prediction.

Encoder-decoder CNN:

The encoder-decoder CNN is typically used for semantic segmentation. In, lane detection was investigated in a transfer learning framework. The end-to-end encoder-decoder network is built on the basis of road scene object segmentation task, trained on Image Net. In, a network called LaneNet was proposed. LaneNet is built on SegNet, but has two decoders. One decoder is a segmentation branch, detecting lanes in a binary mask. The other is an embedding branch, segmenting the road. As the output of the network is generally a feature map, clustering and curve-fitting algorithms were then required to produce the final results In , real-time road marking segmentation was exploited in a condition of lack of large amount of labeled data for training.

Parameter analysis:

There are mainly two parameters that may influence the performance of the proposed methods. One is the number of frames used as the input of the networks; the other is the stride for sampling. These two parameters determine the total range between the first and the last frame together. While given more frames as the input for the proposed networks, the models can generate the prediction maps with more additional information, which may be helpful to the final results. However, in other hand, if too many previous frames are used, the outcome may be not good as lane situations in far former frames are sometimes significantly different from the current frame. Thus, we firstly analyze the influence caused by the number of images in the input sequence. We set the number of images in the sequence from to and compare the results at these five different values with the sampling strides .

CODING AND ITS IMPLEMENTATION

Main.py

```
# import libaries
import matplotlib.pyplot as plt
import cv2
import os, glob
import numpy as np
from moviepy.editor import VideoFileClip

#matplotlib inline
#config InlineBackend.figure_format = 'retina'
# used to display images

def show_images(images, cmap=None):
   cols = 2
   rows = (len(images)+1)m//cols

   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()
# plot the test images

test_images = [plt.imread(path) for path in glob.glob('test_images/*.jpg')]
```

```
show_images(test_images)
# selecting only yellow and white colors in the images using the RGB
channels

def select_rgb_white_yellow(image):
   # white color mask
   lower = np.uint8([200, 200, 200])
   upper = np.uint8([255, 255, 255])
   white_mask = cv2.inRange(image, lower, upper)
   # yellow color mask
   lower = np.uint8([190, 190,   0])
   upper = np.uint8([255, 255, 255])
   yellow_mask = cv2.inRange(image, lower, upper)
   # combine the mask
   mask = cv2.bitwise_or(white_mask, yellow_mask)
   masked = cv2.bitwise_and(image, image, mask = mask)
   return masked
# RGB images are converted into HSV color space

def convert_hsv(image):
   return cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
   show_images(list(map(convert_hsv, test_images)))
# RGB images are converted into HSL color space
```

```
def convert_hls(image):
   return cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
   show_images(list(map(convert_hsl, test_images)))
   show_images(list(map(select_rgb_white_yellow, test_images)))
# filter to select those white and yellow lines

def select_white_yellow(image):
   converted = convert_hls(image)
   # white color mask
   lower = np.uint8([  0, 200,   0])
   upper = np.uint8([255, 255, 255])
   white_mask = cv2.inRange(converted, lower, upper)
   # yellow color mask
   lower = np.uint8([ 10,   0, 100])
   upper = np.uint8([ 40, 255, 255])
   yellow_mask = cv2.inRange(converted, lower, upper)
   # combine the mask
   mask = cv2.bitwise_or(white_mask, yellow_mask)
   return cv2.bitwise_and(image, image, mask = mask)


white_yellow_images = list(map(select_white_yellow, test_images))


show_images(white_yellow_images)
# gray scaling
```

```
def convert_gray_scale(image):
    return cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

gray_images = list(map(convert_gray_scale, white_yellow_images))

show_images(gray_images)
# Gaussian Smoothing

def apply_smoothing(image, kernel_size=15):

    return cv2.GaussianBlur(image, (kernel_size, kernel_size), 0)

blurred_images = list(map(lambda image: apply_smoothing(image),
gray_images))

show_images(blurred_images)
def detect_edges(image, low_threshold=50, high_threshold=150):
    return cv2.Canny(image, low_threshold, high_threshold)

edge_images = list(map(lambda image: detect_edges(image),
blurred_images))

show_images(edge_images)
def filter_region(image, vertices):
    mask = np.zeros_like(image)
```

```
   if len(mask.shape)==2:
       cv2.fillPoly(mask, vertices, 255)
   else:
       cv2.fillPoly(mask, vertices, (255,)*mask.shape[2]) # in case, the input
image has a channel dimension
   return cv2.bitwise_and(image, mask)



def select_region(image):
   # first, define the polygon by vertices
   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]
   # the vertices are an array of polygons (i.e array of arrays) and the data
type must be integer
   vertices = np.array([[bottom_left, top_left, top_right, bottom_right]],
dtype=np.int32)
   return filter_region(image, vertices)



# images showing the region of interest only
roi_images = list(map(select_region, edge_images))
```

```
show_images(roi_images)
def hough_lines(image):

   return cv2.HoughLinesP(image, rho=1, theta=np.pi/180, threshold=20,
minLineLength=20, maxLineGap=300)



list_of_lines = list(map(hough_lines, roi_images))
# draw lines on actutal image

def draw_lines(image, lines, color=[255, 0, 0], thickness=2,
make_copy=True):
   # the lines returned by cv2.HoughLinesP has the shape (-1, 1, 4)
   if make_copy:
       image = np.copy(image) # don't want to modify the original
   for line in lines:
       for x1,y1,x2,y2 in line:
           cv2.line(image, (x1, y1), (x2, y2), color, thickness)
   return image



line_images = []
for image, lines in zip(test_images, list_of_lines):
   line_images.append(draw_lines(image, lines))
```

```
show_images(line_images)
### Averaging and Extrapolating Lines : averaged multiple lines detected for
a lane line

def average_slope_intercept(lines):
   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 x2==x1:
               continue # ignore a vertical line
           slope = (y2-y1)/(x2-x1)
           intercept = y1 - slope*x1
           length = np.sqrt((y2-y1)**2+(x2-x1)**2)
           if slope < 0: # y is reversed in image
               left_lines.append((slope, intercept))
               left_weights.append((length))
           else:
               right_lines.append((slope, intercept))
               right_weights.append((length))

   # add more weight to longer lines
```

```
    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 # (slope, intercept), (slope, intercept)
#   Convert a line represented in slope and intercept into pixel points

def make_line_points(y1, y2, line):

    if line is None:
        return None

    slope, intercept = line

    # make sure everything is integer as cv2.line requires it
    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):
    left_lane, right_lane = average_slope_intercept(lines)
```

```
   y1 = image.shape[0] # bottom of the image
   y2 = y1*0.6         # slightly lower than the middle

   left_line  = make_line_points(y1, y2, left_lane)
   right_line = make_line_points(y1, y2, right_lane)

   return left_line, right_line


def draw_lane_lines(image, lines, color=[255, 0, 0], thickness=20):
   # make a separate image to draw lines and combine with the orignal later
   line_image = np.zeros_like(image)
   for line in lines:
       if line is not None:
           cv2.line(line_image, *line,  color, thickness)
   # image1 * α + image2 * β + λ
   # image1 and image2 must be the same shape.
   return cv2.addWeighted(image, 1.0, line_image, 0.95, 0.0)


lane_images = []
for image, lines in zip(test_images, list_of_lines):
   lane_images.append(draw_lane_lines(image, lane_lines(image, lines)))
```

```
show_images(lane_images)
```

Package used:

CV2

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of

OpenCV. OpenCV's deployed uses span the range from stitching streetview images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

OpenCV Library Modules

Following are the main library modules of the OpenCV library.

Core Functionality

This module covers the basic data structures such as Scalar, Point, Range, etc., that are used to build OpenCV applications. In addition to these, it also includes the multidimensional array Mat, which is used to store the images. In the Java library of OpenCV, this module is included as a package with the name org.opencv.core.

Image Processing

This module covers various image processing operations such as image filtering, geometrical image transformations, color space conversion, histograms, etc. In the Java library of OpenCV, this module is included as a package with the name org.opencv.imgproc.

Video

This module covers the video analysis concepts such as motion estimation, background subtraction, and object tracking. In the Java library of OpenCV, this module is included as a package with the name org.opencv.video.

Video I/O

This module explains the video capturing and video codecs using OpenCV library. In the Java library of OpenCV, this module is included as a package with the name org.opencv.videoio.

calib3d

This module includes algorithms regarding basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence and elements of 3D reconstruction. In the Java library of OpenCV, this module is included as a package with the name org.opencv.calib3d.

features2d

This module includes the concepts of feature detection and description. In the Java library of OpenCV, this module is included as a package with the name org.opencv.features2d.

Objdetect

This module includes the detection of objects and instances of the predefined classes such as faces, eyes, mugs, people, cars, etc. In the Java library of OpenCV, this module is included as a package with the name org.opencv.objdetect.

Highgui

This is an easy-to-use interface with simple UI capabilities. In the Java library of OpenCV, the features of this module is included in two different packages namely, org.opencv.imgcodecs and org.opencv.videoio.

Implementation :

Python

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An Interpreted Language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or java. It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many operating systems. C Python, the reference implementation of Python, is open-source software and has a community-based development model, as do nearly all of its variant implementations. C Python is managed by the non - profit Python Software Foundation. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object -oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Flask:

Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.

Flask is part of the categories of the micro-framework. Micro-framework are normally framework with little to no dependencies to external libraries.

Flask is used for developing web applications using python, implemented on Werkzeug and Jinja2.

Advantages of using Flask framework are:

- There is a built-in development server and a fast debugger provided.
- Lightweight
- Secure cookies are supported.
- Templating using Jinja2.
- Request dispatching using REST.
- Support for unit testing is built-in

The following command installs virtualenv pip install virtualenv

This command needs administrator privileges. Add sudo before pip on

Linux/Mac OS. If you are on Windows, log in as Administrator. On Ubuntu

virtualenv may be installed using its package manager.

Sudo apt-get install virtualenv

Once installed, new virtual environment is created in a folder. mkdir newproj

cd newproj virtualenv venv

To activate corresponding environment, on Linux/OS X, use the following −

venv/bin/activate

On Windows, following can be used venv\scripts\activate

We are now ready to install django in this environment.

pip install flask

SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail. Test objectives

• All field entries must work properly.
• Pages must be activated from the identified link.
• The entry screen, messages and responses must not be delayed. Features to be tested
• Verify that the entries are of the correct format
• No duplicate entries should be allowed.

• All links should take the user to the correct page.

Integration Testing

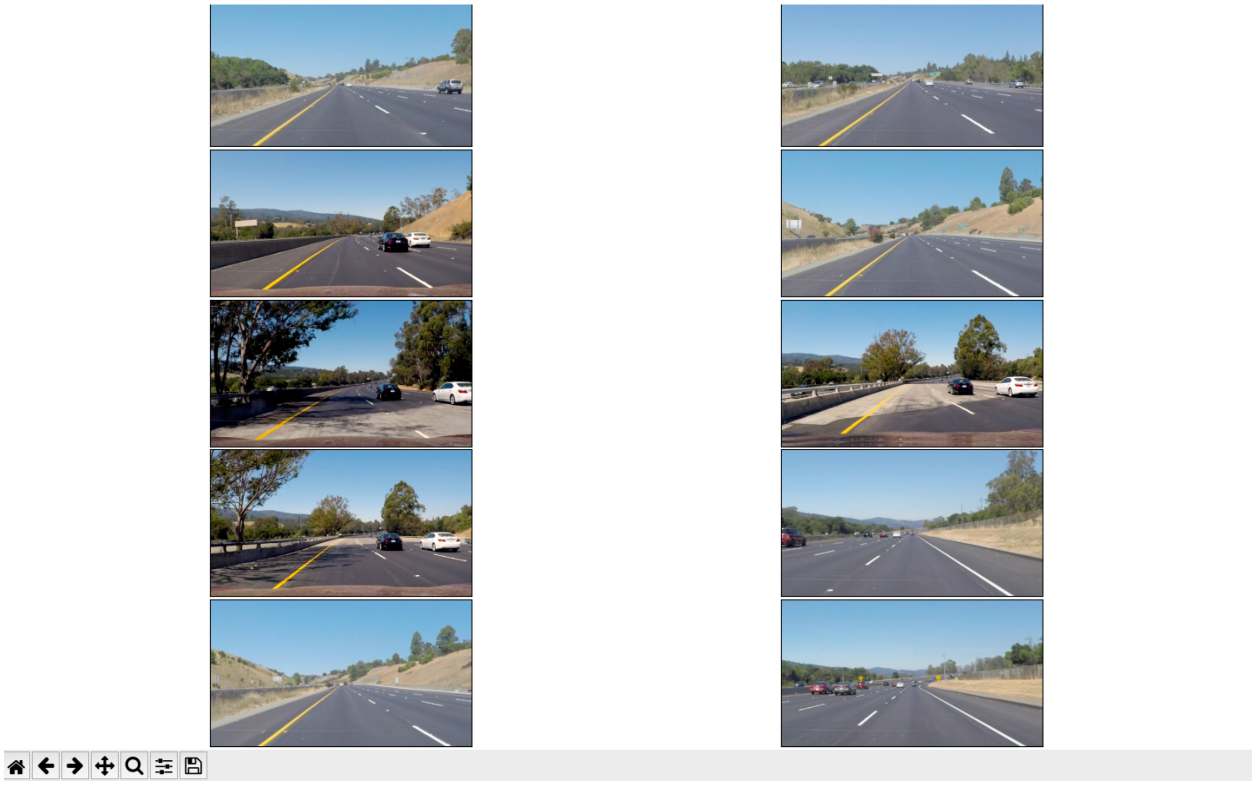
Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, e.g., components in a software system or – one step up – software applications at the company level – interact without error.

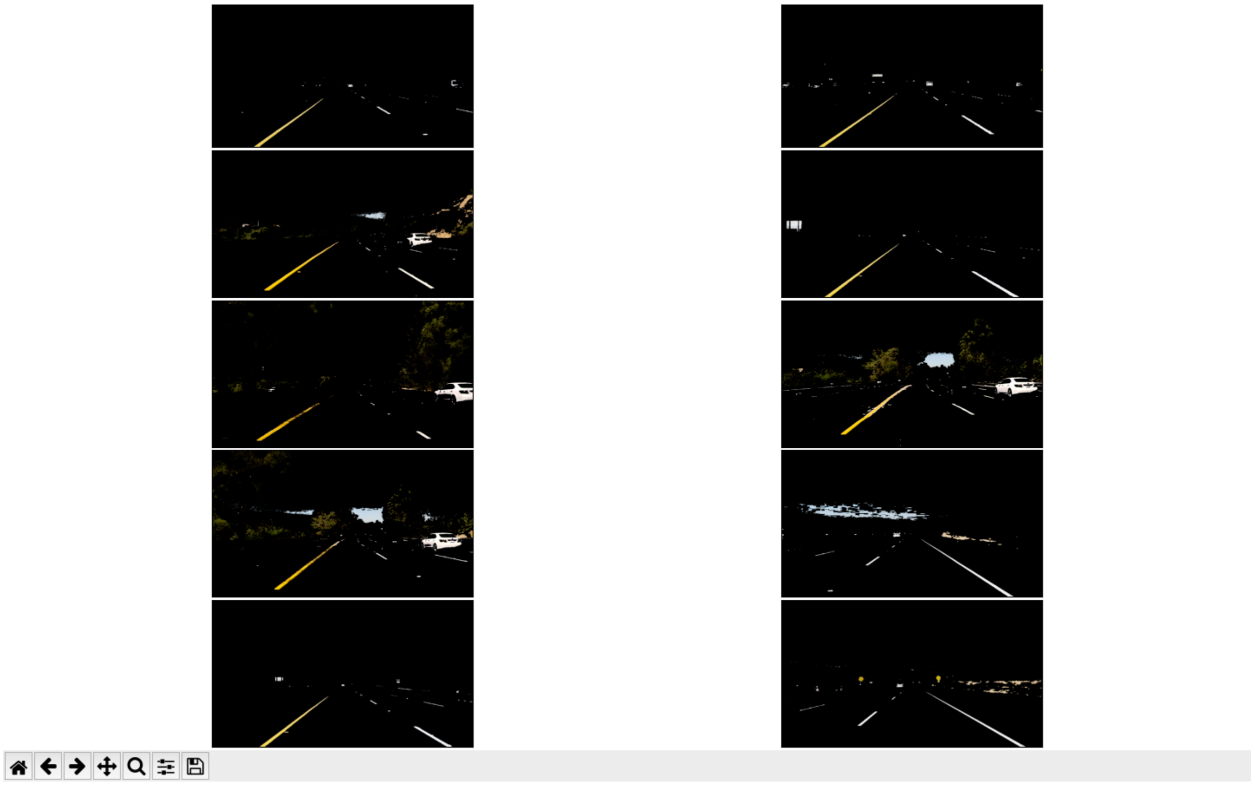
Test Results
All the test cases mentioned above passed successfully. No defects encountered. Acceptance Testing
User Acceptance
Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

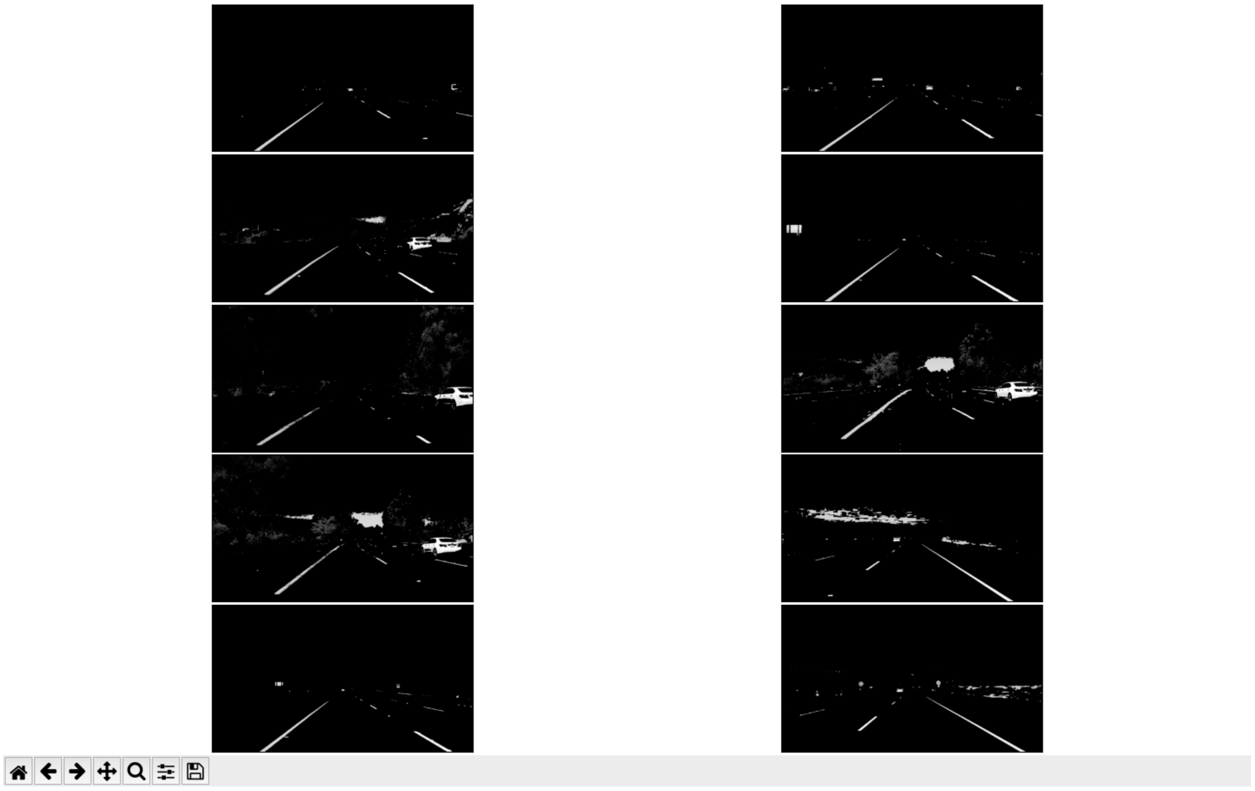
Test Case:

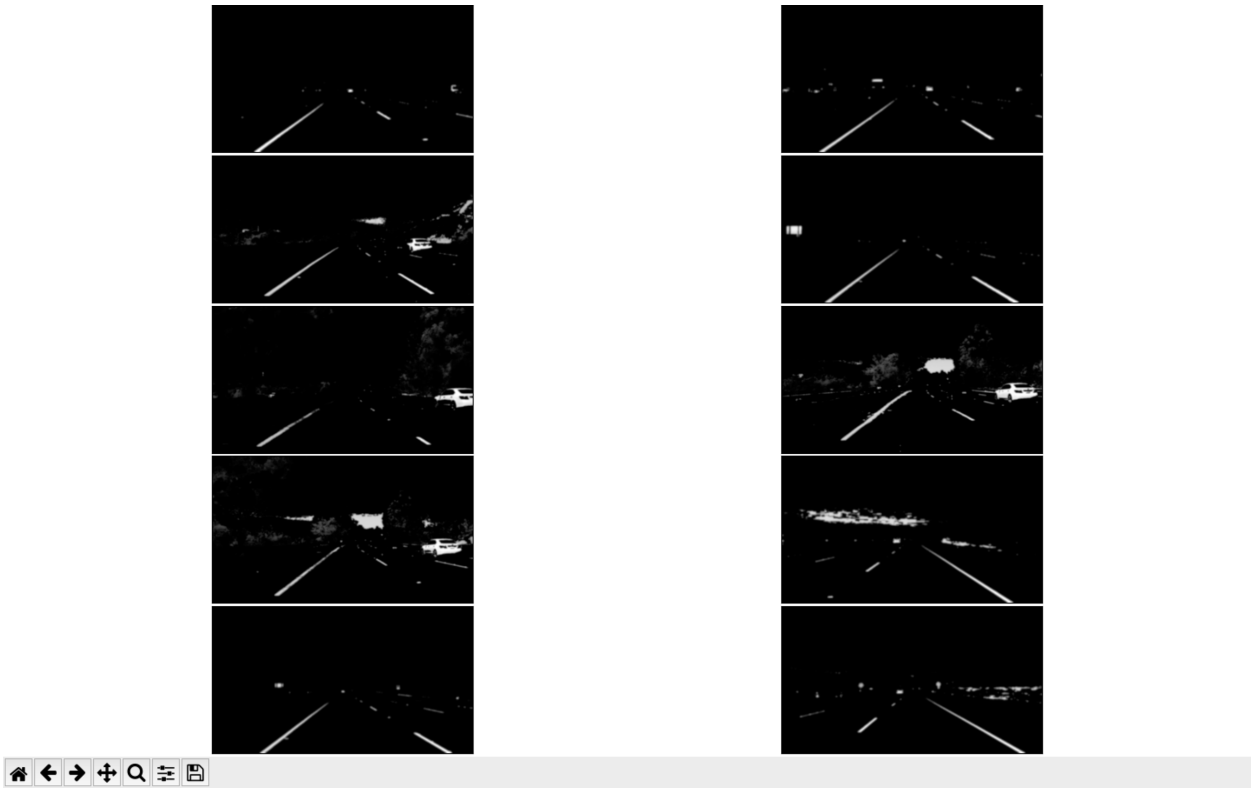
S.No	Test Case	Expected Result	Result	Remarks(If Fails)
1	Upload road vehicle images	Lanes will be highlighted	Pass	Image cannot be detected or Error

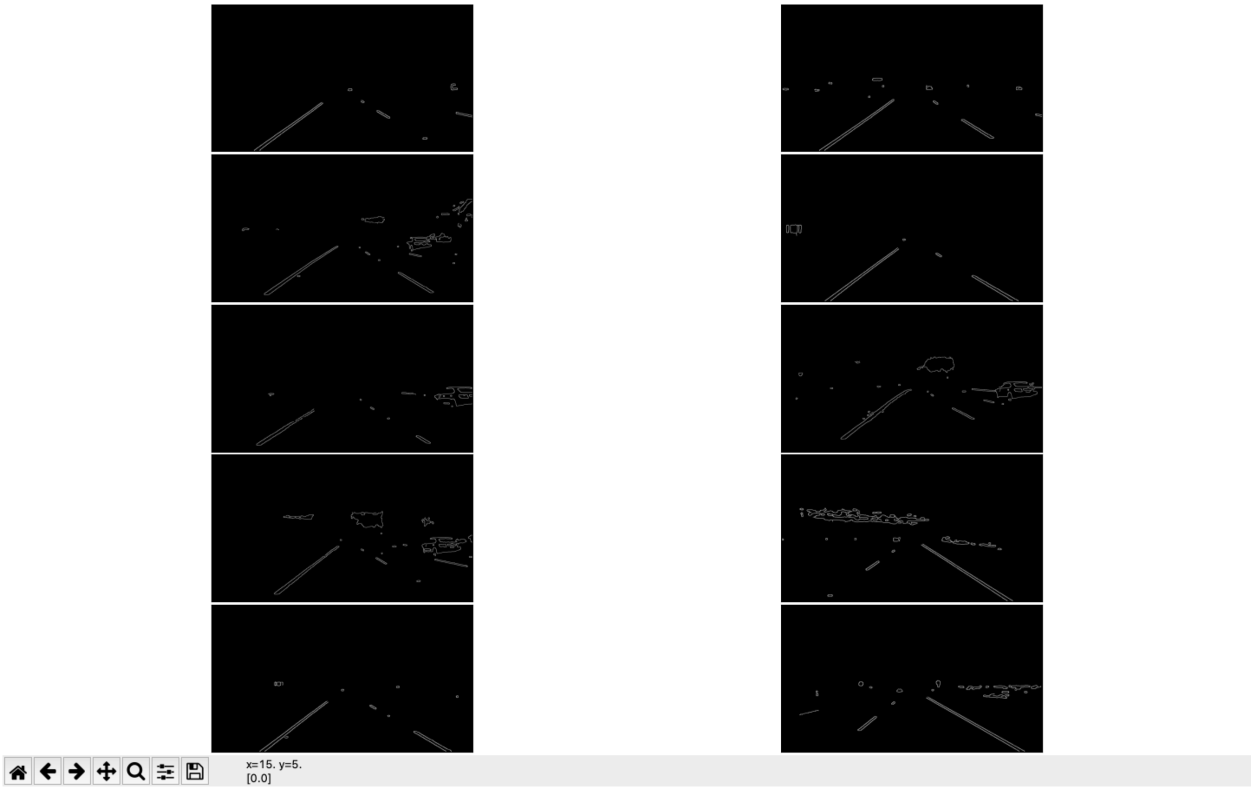
Screen shots:



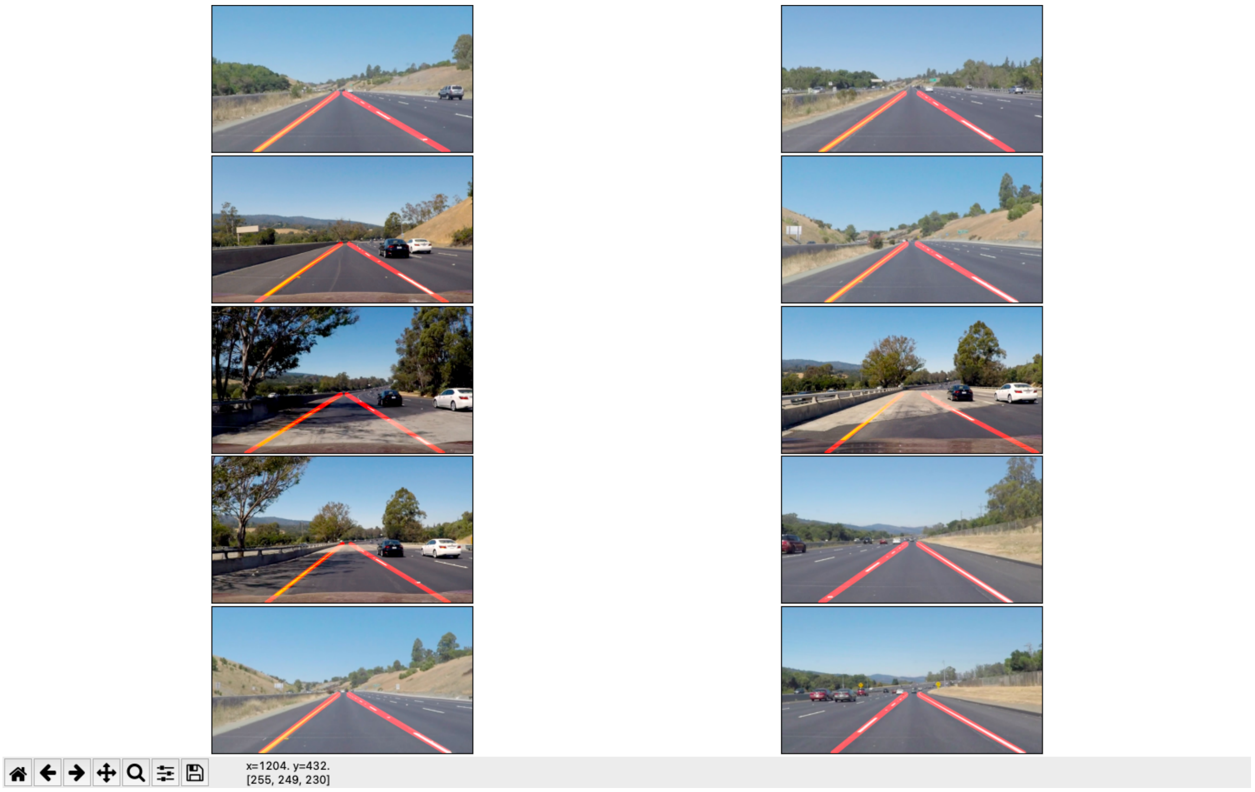








```
x=15. y=5.
[0.0]
```



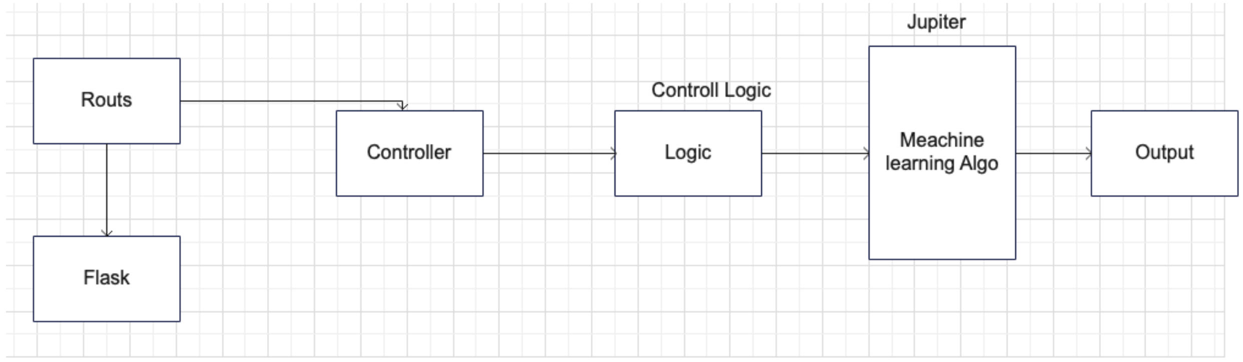
```
x=1204. y=432.
[255, 249, 230]
```

Conclution:

Compared with baseline architectures which use one sin
gle image as input, the proposed architecture achieved signifificantly better results, which verifified the effectiveness of
using multiple continuous frames as input. Meanwhile, the experimental results demonstrated the advantages of Con
vLSTM over FcLSTM in sequential feature learning and target-information prediction in the context of lane detection. Compared with other models, the proposed models showed higher performance with relatively higher precision, recall, and accuracy values. In addition, the proposed models were tested on a dataset with very challenging driving scenes to check the robustness. The results showed that the proposed models can stably detect the lanes in diverse situations and can well avoid false recognitions. In parameter analysis, longer sequence of inputs were found to make improvement on the performance, which further justified the strategy that multiple frames are more helpful than one single image for lane detection.

References
copy from Base paper

System Architecture

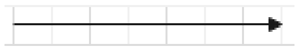


If the user clicks on the Prediction data the data will get preprocessed and display the Crowd.

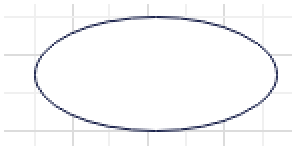
Data Flow Diagrams

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called a data flow graph or bubble chart. The Basic Notation used to create a DFD's are as follows:

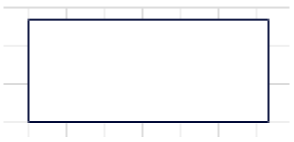
1. Dataflow: Data move in a specific direction from an origin to a destination.

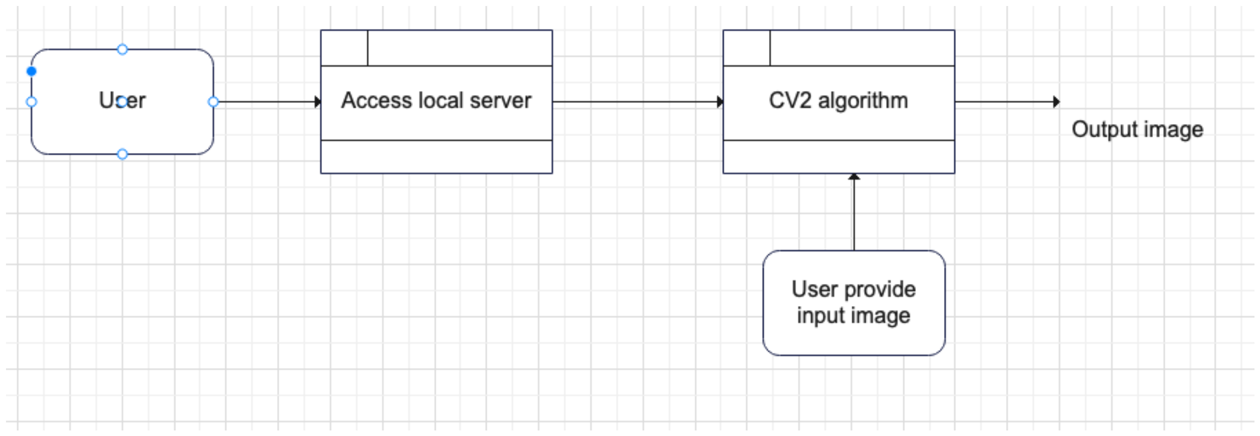


2.Process: People, procedures, or devices that use or produce (Transform) Data. The physical component is not identified.



3.Source: External sources or destination of data, which may be People, programs.

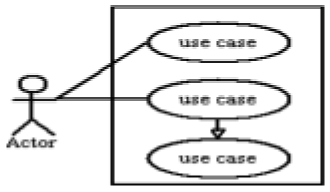




What is UML diagram?
There are several types of UML diagrams and each one of them serves a different purpose regardless of whether it is being designed before the implementation or after (as part of documentation).

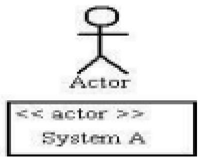
The two broadest categories that encompass all other types are:
1. Behavioral UML diagram and 2. Structural UML diagram.

What is a UML Use Case Diagram?
Use case diagrams model the functionality of a system using actors and use cases. Use cases are services or functions provide by the system to its users. Basic Use Case Diagram Symbols and Notations System Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.

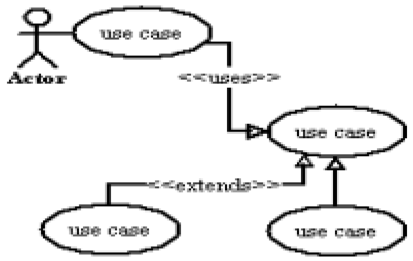


Use Case Draw the use case using ovals.

Label with verbs that represent the system's functions.
Actors Actors are the users of a system.
When one system is the actor of another system, label the actor system with the actor stereotype.



Relationships Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform task.

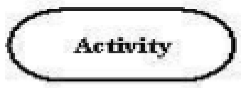


What is a UML activity diagram?
Activity Diagram

An activity diagram illustrates the dynamic nature of a system by modelling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. Because an activity diagram is a special king of state chart diagram, it uses some of the same modelling conventions.

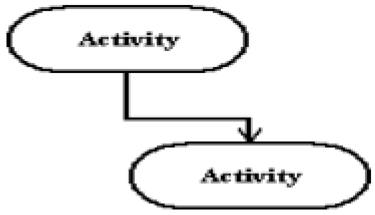
Basic Activity Diagram Symbols and Notations Action states

Action states represent the non-interruptible actions of objects. You can grow an action state in Smart Draw using a rectangle with rounded corners.



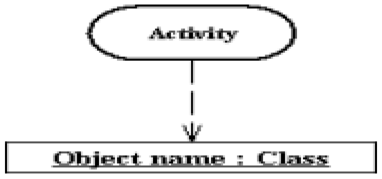
Action Flow

Action flow arrows illustrate the relationships among action states.



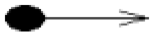
Object Flow

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.



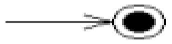
Initial State

A filled circle followed by an arrow represents the initial action state.



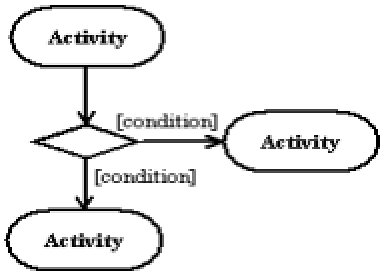
Final State

An arrow pointing to a filled circle nested inside another circle represents the final action state.



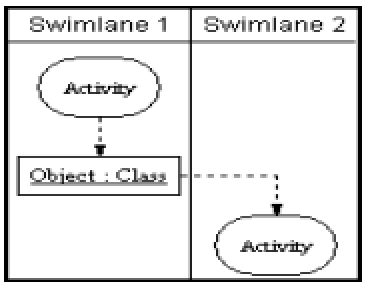
Branching

A gaining represents a decision with alternate paths. The outgoing alternates should be labeled with a cognition or guard expression.



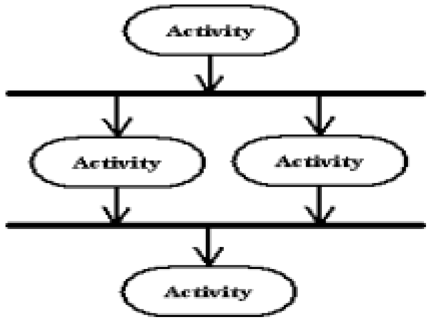
Swim lanes

Swim lanes group related activities into one column.



Synchronization

A synchronization bar helps illustrate parallel transitions. Synchronization is also called forking and joining.

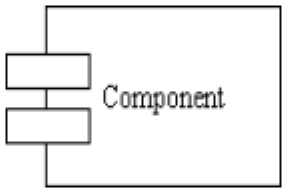


What is a UML Component Diagram?

A component diagram describes the organization of the physical components in a system.

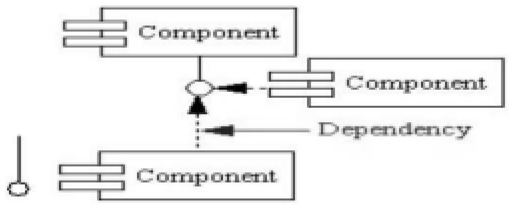
Component

A component is a physical building block of the system Learn how to resize grouped objects like components.



Interface

An interface describes a group of operations used or created by components.

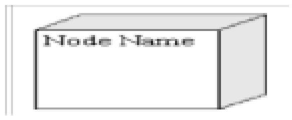


Dependencies

Draw dependencies among components using gashes arrows. Learn about line styles in Smart Draw.

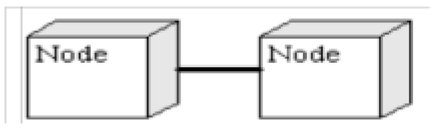
Component

A node is a physical resource that executes code components. Learn how to resize grouped objects like nodes.



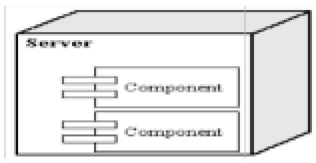
Association

Association refers to a physical connection between nodes, such as Ethernet. Learn how to connect two nodes.



Components and Nodes

Place components inside the node that deploys them.



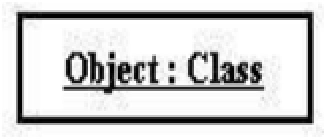
What is a UML sequence diagram?

Sequence Diagram

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time.

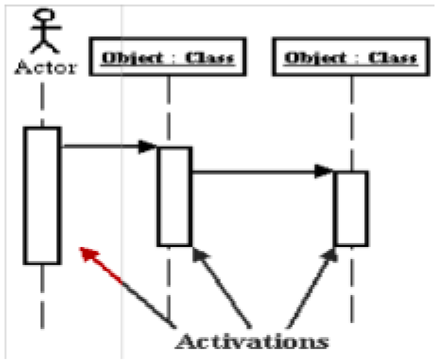
Basic Sequence Diagram Symbols and Notations Class roles

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.



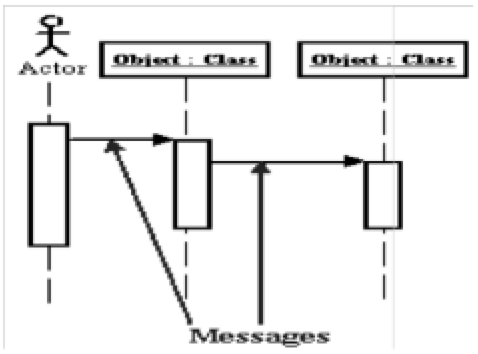
Activation

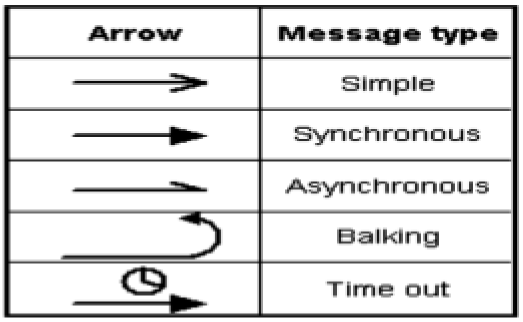
Activation boxes represent the time an object needs to complete a task



Messages

Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks.

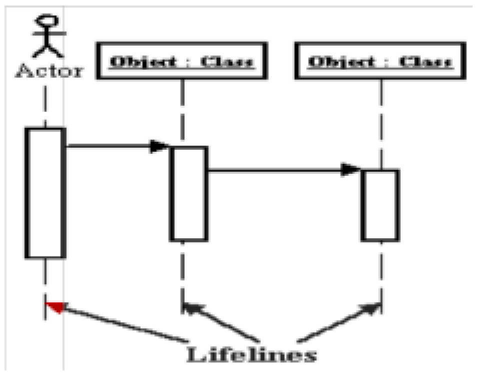




Arrow	Message type
→	Simple
→	Synchronous
⇀	Asynchronous
↩	Balking
◷ →	Time out

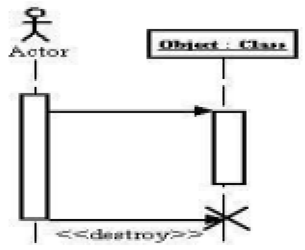
Lifelines

Lifelines are vertical gashes lines that indicate the object's presence over time.



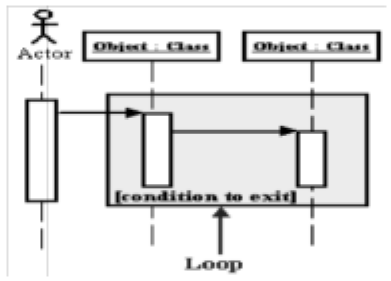
Destroying

Objects Objects can be terminated early using an arrow labelled "<< destroy >>" that points to an X.



Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the cognition for exiting the loop at the bottom left corner in square brackets.



UML Diagrams

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

There are several types of UML diagrams and each one of them serves a different purpose regardless of whether it is being designed before the implementation or after (as part of documentation). UML has a direct relation with object-oriented analysis and design. After some standardization, UML has become an OMG standard.

The two broadest categories that encompass all other types are:

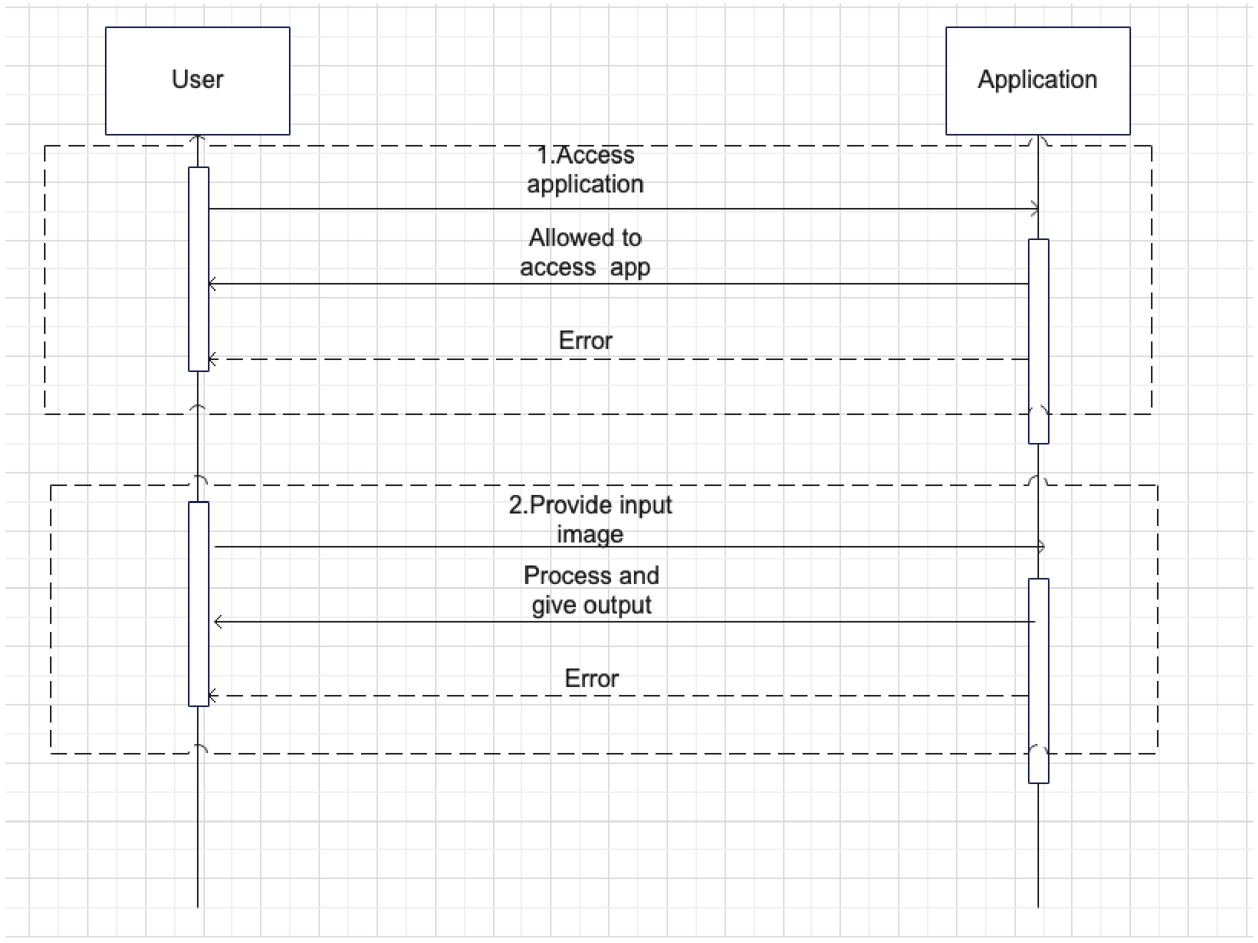
1. Behavioral UML diagram and
2. Structural UML diagram.

As the name suggests, some UML diagrams try to analyze and depict the structure of a system or process, whereas others describe the behavior of the system, its actors, and its building components. The different types are broken down as follows:

1. Sequence diagram
2. Use case Diagram
3. Activity diagram
4. Class diagram

1. Sequence diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e., the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.



List of actions User:

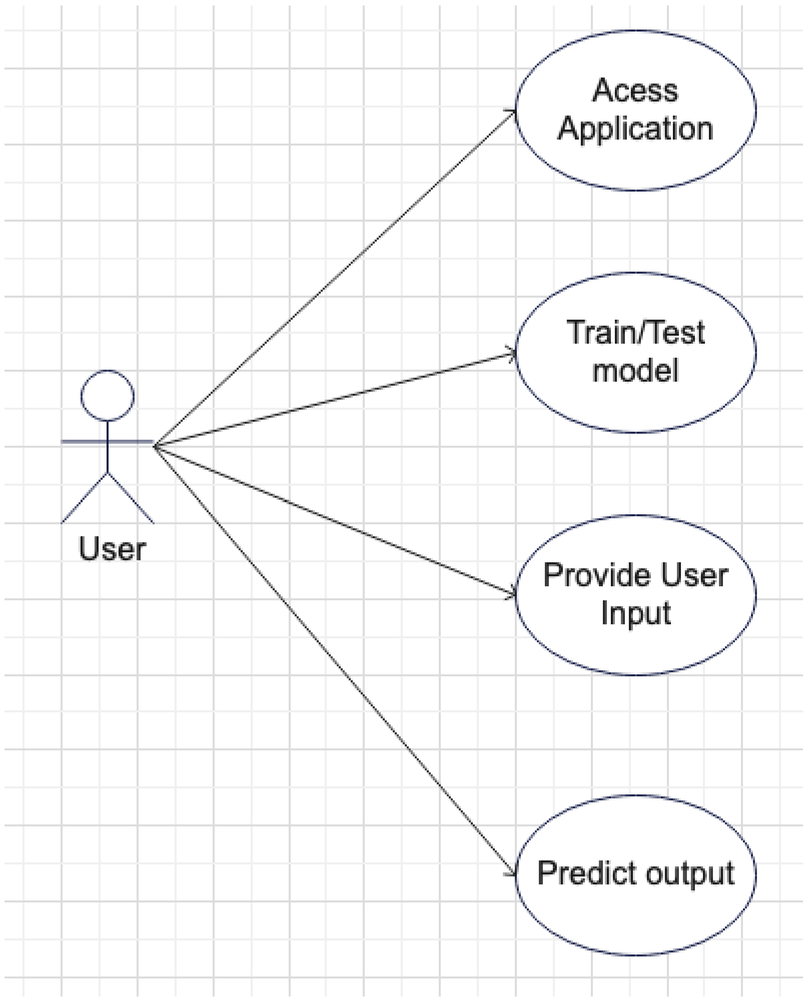
User Provide input text to be summarized. The text will be undergoing a series of texts as mentioned in the DFL diagram.

Result:

As per user enters the data it will give the text will be summarized.

2. Use case Diagram

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use case in which the user is involved. A use case diagram is used to structure the behavior thing in a model. The use cases are represented by either circles or ellipses.



Actor:

User

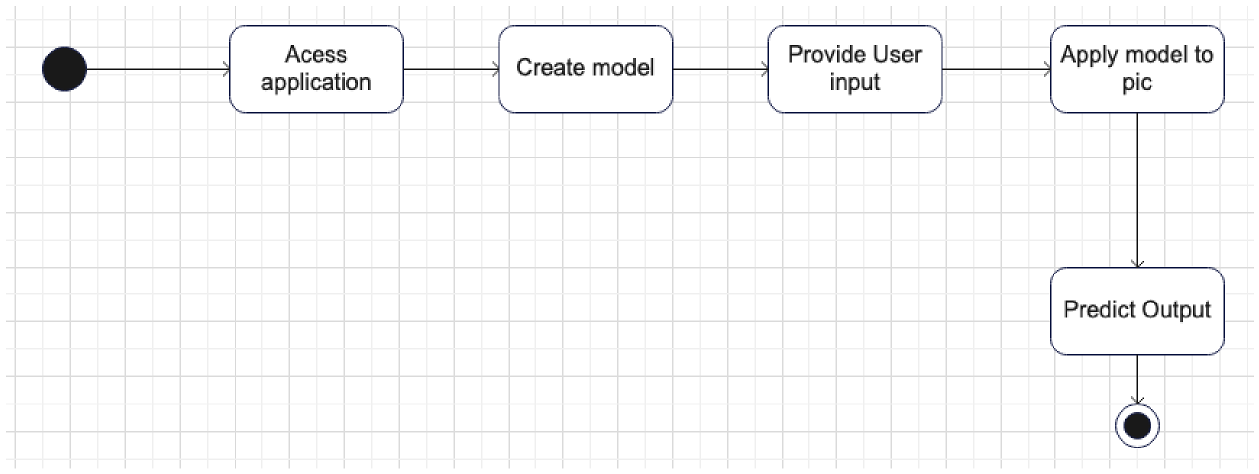
System

Use case:

User needs to give the data. Then the System will give the results. System Consist of data set that will preprocess the data and then split the data and apply the train and test the data then it will predict the results.

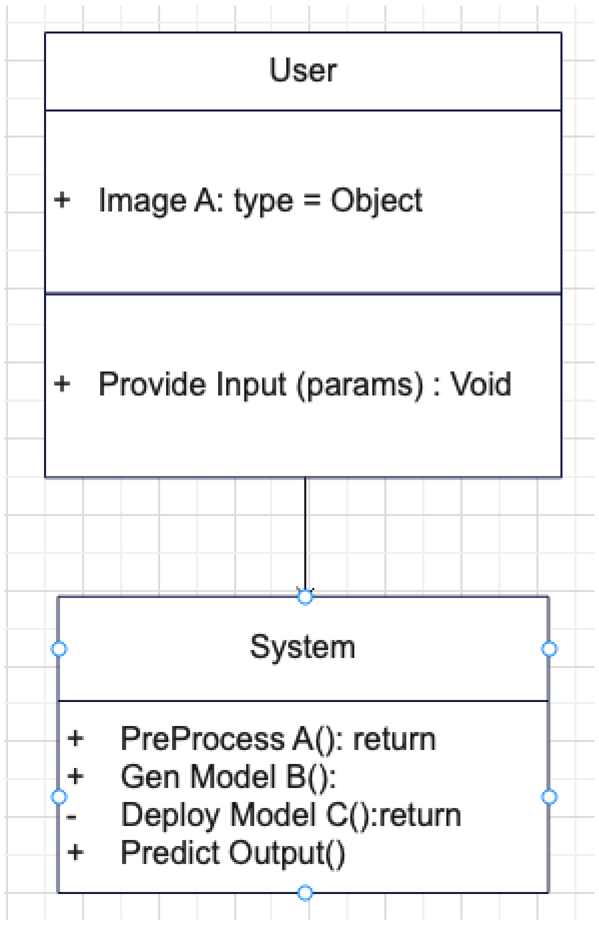
3. Activity diagram

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all types of flow control by using different elements such as fork, join, etc.



4. CLASS DIAGRAM

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.


User
+ Image A: type = Object
+ Provide Input (params) : Void
System
+ PreProcess A(): return
+ Gen Model B():
- Deploy Model C():return
+ Predict Output()