Driving Assistance Platform for Self Driving Vehicles

Project report submitted in partial fulfilment of the requirement for the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

By

Sarthak Srivastava 161115

UNDER THE GUIDANCE OF

Mr. Pardeep Garg

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,WAKNAGHAT

May 2020

TABLE OF CONTENTS

DECLARATION

We hereby declare that the work reported in the B. Tech Project Report entitled **Driving Assistance Platform for Self-Driving Vehicles"** submitted at **Jaypee University of Information Technology, Waknaghat, India** is an authentic record of our work carried out under the supervision of **Mr. Pardeep Garg.** We have not submitted this work elsewhere for any other degree or diploma.

Sarthak Srivastava

161115

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Mr. Pardeep Garg

Date:

Head of the Department/Project Coordinator

ACKNOWLEDGEMENT

We would like to express my gratitude to our supervisor **Mr. Pardeep Garg, Assistant Professor**, Department of Electronics and Communications Engineering of Jaypee University, Waknaghat for the useful guidance, remarks and engagement in the learning's process of this bachelor project. **Mr. Pardeep Garg** consistently allows this project to be our own work, but always steers us in the right direction whenever he thinks we need it.

We are highly grateful to **all the technical staff of Jaypee University of Information Technology** for their guidance right from the very beginning. They actually laid the ground for conceptual understanding of technologies we used in the project. This accomplishment would not have been possible without them.

Thank you.

By:

Sarthak Srivastava 161115

LIST OF ACRONYMS AND ABBREVIATIONS

IEEE	INSTITUE OF ELECTICAL AND ELECTRONICS ENGINEERS
FCN	FULLY CONVOLUTIONAL NUERAL NETWORK
CNN	CONVOLUTIONAL NEURAL NETWORKS
IDD	INDIAN DRIVING DATASET
CV	COMPUTER VISION
GPU	GRAPHIC PROCESSING UNIT
LIDAR	LIGHT DETECTION AND RANGING
PCB	PRINTED CIRCUIT BOARD
PC	PERSONAL COMPUTER
USB	UNIVERSAL SERIAL BUS

HOG	HISTOGRAM OF ORIENTED GRADIENTS
CVPR	CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION
UI	USER INTERFACE
IDE	INTEGRATED DEVELOPMENT ENVIROMENT
HLS	HTTP LIVE STREAMING
RGB	RED GREEN BLUE

LIST OF SYMBOLS

yij	Mapped output in i,j space.
fks	Function of K measuring in S space.
δ	Delta function (Partial Derivative)
$\circ$	Dot Product
$\sum$	Summation
$f_{ij}=0$	Initial Values of the function in i,j space.
(x,y)	X,Y coordinates in 2-D space.
$\int$	Integration
τ	Tau (Small shift in time)
*	Convolution

LIST OF FIGURES

ABSTRACT

The project is about development of a prototype of **"Driving Assistance Platform for Self-Driving Vehicles".** In this ever-evolving world, where evolution of a Self-Driving Vehicle has become an essential part of new innovations and technology; we sometimes forget to focus on the safety. This is why; technological advancements are taking place in this particular field, to help us live our lives with ease, not worrying about the nitty-gritty of existence. The following project is our humble take on improving quality of Humans and vehicles interaction.

How many times has it happened that we have seen vehicles colliding to each other due to human errors and carelessness? The best of us commit mistakes, *to err is human*, they say. Imagine a device that assists you in interacting with the on-board computer system on Vehicle, so that our mistakes can be minimized and also there is an efficient control system which is based on real life practical scenarios. This is where our assistance platform comes in. From Dashboard camera based front pedestrian detection and vehicular collision avoidance, to lane departure warning, to tailgate warning for any vehicle approaching from behind, it can do it all. You can also get live feed from all the cameras to the dashboard screen.

Worried about reaching home safe? Self-Driving Vehicles are here as an autonomous Robotic Cr capable of sensing its environment and navigating without human input. Alert systems can also be set, so that you're notified of any unfortunate circumstance. Basically, it is an interactive assistance platform for vehicles with added safety.

CHAPTER 1
INTRODUCTION

A Self Driving Assistance Platform enables a car to sense its environment and without any human input. It's able to detect other vehicles, lanes, traffic lights, stop signs and people on the road and operate accordingly. It uses techniques such as radar, GPS and Computer Vision. Driving assistance platforms have been developed for few years now using all sorts of technologies and algorithms. Our Idea with this project is to approach this project with Computer Vision techniques and then try to improve result's accuracy, efficiency, reliability and sustainability using Machine Learning algorithms.

1.1 Background of Self Driving cars

The Self - Driving car idea as a whole has been introduced since the early 90s way before Google started researching on this and now their cars can be seen driving on the streets of San Francisco. Since 2013 few states in USA have approved testing of self driven cars. The idea has always been that if all the cars are robotic they can communicate better with each other as well and ensure safety of the passenger and increase driving efficiency.

The first idea of a self driving vehicle was introduced in New York World's fair by the General motors group in the Futurama exhibit. They envisioned a highway system that would support self driving vehicles to transport people from one place to another.

Figure 1.1: The Futurama, GM's Highway System [1]

Back 1920s almost every Automobile car company that envisioned Self driving cars based on Radio control Technology. In 1925 the Houdina Radio Control Company drove the Lemurian wonder 825 chandler in New York city without any driver, instead they had an advanced radio unit and antenna and another car behind them where the actual driver was controlling it using remote control. A year later the Phantom Motor Car Company operated by the Akun Motor company made a Car that could start it's own motor, shifted it's on clutch, shifted it's own gears and twisted it's own steering wheel. The GM's vision was to control Cars using radio but propelled with electromagnets that were buried in the highways. They claimed to ensure safe distancing between cars by "Automatic Radio Control".

In 1939 a Non-radio approach was suggested which would be based on LASERs. The electric eye would project a concentrated light beam which would be reflected back at the car with mirrors that would enable it to detect them. These electric eyes were photo detectors. So in theory if the projected light by the car was obstructed then it would tell the car to turn or break.

In 1950s at the GM's Motorama a film was presented which was that in the near future as in 1976 they imagined a series of control towers that route and control cars remotely on special electronic highway lanes. In this you would call the tower, tell them your destination and the people in the tower would set your lane, match your speed and have electronic control over your car. The system was discarded a year later because it was too expensive.

In 1980s the DARPA (Defense Advanced Research Project Agency) made ALV which was Autonomous land vehicle which was the first Self Driving Car. It used Lidar and Computer Vision to travel around 600 meters by itself.

Later companies like Mercedes Benz and Google researching and experimenting in this using Machine learning Algorithms.

1.2 Motivation

The motivation behind this project is that a self driving assistance platform can help reduce road accidents and traffic issues that occur every day. This platform can detect and recognize objects and lanes on the road and suggest directions and functions to the user while driving in real time. Just like a GPS that suggests driver to take a certain route this system will suggest the driver to stay in a certain lane or when to stop detecting a stop sign or detect vehicles in front of them.

Autonomous vehicles can bring significant reduction in road accidents because they are not prone to make humanly mistakes while driving and they are free from human errors in every situation. For instance the self driven car can park itself without any problem. A self driving system does not need to sleep during long hours of journey on the road whether it is day or night it needs no rest. A computer is much less likely to get into an accident due to road rage as well.

1.3 Different levels of Autonomy

Level 0: No Automation

The human driver has full oversight of the vehicle, despite the fact that there are redesigns and innovative development in the vehicle, the individual in the driver's seat is the person who responds to each circumstance and handles it.

Level 1: Driver Assistance

The majority of the tasks are dealt with by the driver itself. A few tasks be that as it may, such as slowing down, can be overseen naturally by the vehicle itself through the information recovered from the street conditions.

Level 2: Partial Automation

In Partial automation the user has control over most of the functionality of the car. Only some tasks like how fast the car should be or if the car should be slower in certain areas are automated. There is always a certain extent of speed on which these features are to be used. For Example the new Tesla model comes with a Auto Pilot Feature which drive easily on the highway by lane detection and vehicle detection.

Level 3: Conditional Automation

Most of the driving tasks are done by the computer but there's always an option to switch back and take control by the driver.

Level 4: High Automation

In this level however there is a directing wheel, gas and brake pedal, and an apparatus move, yet the entirety of the driving is robotized and there is additionally a safeguard, should the driver neglect to connect with when the framework requests their extravagance and help.

Level 5: Full Automation

This level readies the vehicle to deal with street conditions and circumstances that can and would emerge during the ride in this manner making it completely mechanized. The driver is thought to be a traveler and their help is neither mentioned nor required.

As it tends to be seen, the last three levels move from the negligible mechanical driver help – the vehicle has certain improvements to make the drive smoother and more secure, into a territory where the vehicle is currently really observing nature and responding naturally.

1.4 Objective

•**Reduced vehicle crashes:** In India road rage and vehicle crashes has consistently been a significant worry of traffic police. Many individuals lose their lives because of street mishaps consistently. This framework will help diminish that number fundamentally and make it a simple and safe excursion. No one will jeopardize others driving while inebriated.

•**No requirement for paid drivers:** Suppose in the event that the vehicle has a paid driver, at that point whoever is paying the driver spares the expense of his pay. This may be a serious deal with long stretch trucking firms.

•**Reduces congested roads:** Full time auto-route with ongoing connecting with traffic, which would make it steady, safe, and maintain a strategic distance from car influxes. In doing as such, traffic will move all the more easily making and we would arrive at our goal rapidly with less utilization of fuel.

•**Parking:** They drop you off and park themselves that would spare a great deal of time. No all the more stopping problems.

1.5 The scope of Self Driving cars in India

In India road accidents and traffic jams are more than most of the countries. People lose their life, their time, their money every day because of this. India demands this system more than we realize and admit. Suppose you could reach your destination safely and in 40% less duration without any driver and without any effort.

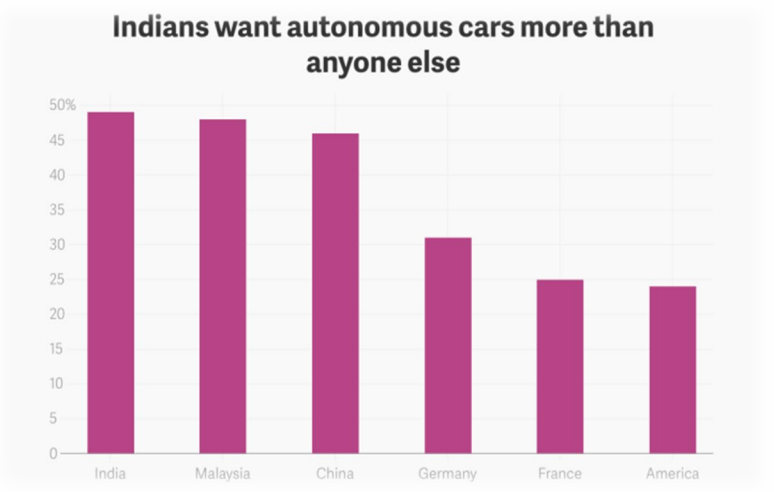


Figure 1.2: Worldwide demand of autonomous cars [2]

As we can see from the Figure above, the demand for autonomous vehicle technology is high even in India. The requirement of autonomous cars in India, which is challenging in a country like India, since they are infamous for being occupied by traffic leading highly congested roads. The number of road accidents and deaths due to them is skyrocketing at an alarming rate. With the significant growth in the field of Machine Learning and Internet of Things, it is safe to say it will pave the way for self-driven vehicles. In India, Ola and many of the start-ups are experimenting with parts of autonomous car technology to come up with a full- fledged autonomous vehicle.

1.6 Hardware Specifications

1. **Raspberry Pi** - The Raspberry Pi 3 is the most important hardware requirement of our project. It's a development board which can be used as a computer in our project. It has very high processing power as compared to other microcontrollers. It has a lot of connectivity support such as Wifi, USB, Ethernet, HDMI, Bluetooth, etc. It's compact size and wide range is very helpful to us to mount it on a car transmit images and video as input to our system. The Raspberry Pi 3 is shown below in Figure 1.3.

Figure 1.3: Raspberry Pi [3]

2. **Raspberry Pi Camera module:** The Raspberry Pi Camera Module V2 is used to record High definition videos and pictures. The camera module is compatible with Raspberry Pi and we are using it for capturing photographs and sending them to our platform for lane detection. It has a 15cm cable to it and it connects with the CSI port on the Raspberry Pi.

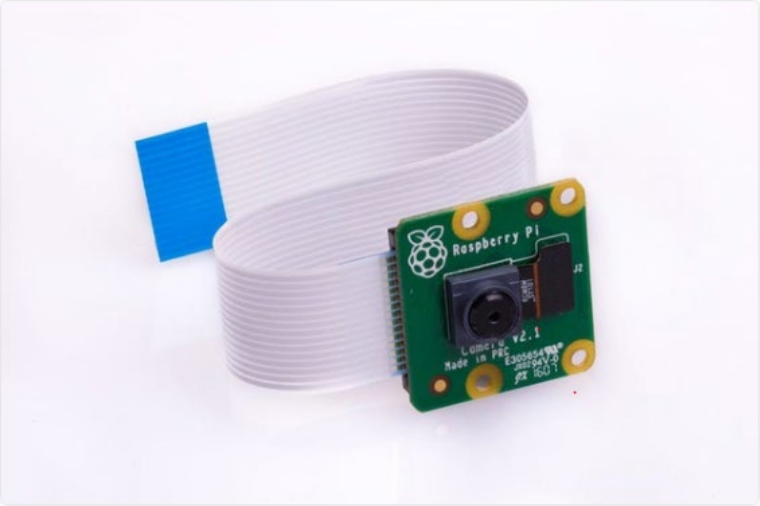


Figure 1.4: Raspberry Pi Camera Module [4]

3. **USB Cable and micro SD card:** We required at least 30cm long USB cable to charge the Raspberry Pi constantly on one end and connected to the computer on the other end. The standard Micro SD card is used to install Raspbian OS on the Raspberry Pi and store images that are captured by the camera module to transmit it to the computer using WIFI connectivity.

Figure 1.5: USB 2.0 cable [5]

4. **RC Car:** Remote controlled car was required to test the software by driving it inside lanes and testing and demonstrating the results. The range of the car should be up to 5 meters at least and considerable battery backup.

5. **Breadboard:** Standard (48 x 35 x 10mm) breadboard for connections to the Raspberry Pi and mounting all the apparatus on top of the RC car.

Figure 1.6: Bread board [6]

1.7 Software Used

1. **Python IDE** - Python IDE is an environment that enables us to develop and run codes in python language. It is supports other software like Spyder, Jupyter, etc. Since most of our project is Python based we require an integrated development Environment that supports us. The Python version we are using is Python 3.6.0 which supports the libraries that we are going to use in our project.

2. **Raspbian** - Raspberry Pi requires a light weight OS to perform operations, for this purpose we have installed Raspbian OS into our microcontroller to enable it's wide range complex functions. Raspbian is like an interface between our computers and Raspberrry Pi. It is based on Linux and it is being rolled out by Raspberry pi organization itself. The version we are using has kernel version 4.19.

3. **Rufus** -Rufus is required because we need to use a memory card as a flash drive to install Raspbian into our Raspberry Pi. Rufus formats the memory card to be able to use as a bootable storage device in our case.

4. **OpenCV Library -**OpenCV-Python is a library of Python bindings designed to solve computer vision problems. Python is a general-purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Uses models to recognize face and helps python in accessing the web camera. Uses stream of images to make a video by combining them.

5. **Anaconda Navigator:** Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository.

6. **Jupyter :** Jupyter is an extensible environment for interactive and reproducible computing, web based notebook. It provides us Editable and readable docs for describing data analysis.

7. **Spyder:** Spyder is a powerful scientific environment written in Python. It enables us to write Python code and offers a unique advanced editing technique with analysis, debugging and profiling functionality with data exploration and interactive execution.

CHAPTER 2

LANE DETECTION AND LANE DEPARTURE WARNING

Lane detection is one of the main projects that independent vehicle-based projects require to be executed in light of the fact that it is the essential guideline of the Autonomous Vehicles. We began with Lane detection. Watching lane lines featured by your calculation as the vehicle moves is very crucial.. Recognizing lane lines is for sure an urgent assignment. It gives parallel limits on the development of vehicle (which will be security basic if vehicle is proceeding onward a side lane), gives thought regarding arch of the street and what amount veered off the vehicle is from the focal point of the lane. Lane detection depends just on camera pictures. We apply computer vision methods to process picture and give lane markings as the yield.

A straightforward detection of lanes can be executed by shading thresholding. In this method, limit is applied over the estimations of shading channels. For example, to identify white lane markings, a suitable edge for RGB shading channel would be [200, 200, 200]. Pixels that have channel esteems over the limit relate to white articles in the picture. Since we are just inspired by white articles in the zone of intrigue, we apply a cover and channel out undesirable detections. Detections in the territory of intrigue can compare to any white item for instance, a white vehicle. This gives us a simple lane detection strategy which possibly works on the off chance that we know about the lane shading in advance.

2.1 Camera Calibration

Camera calibration is the way toward assessing parameters of the camera utilizing pictures of an extraordinary calibration design. The parameters incorporate camera inherent, mutilation coefficients, and camera extraneous. 3-D vision is the way toward reproducing a 3-D scene from at least two perspectives on the scene

2.1.1 Calibration Matrix

Calibration Matrix or projection network is a grid which depicts the mapping of a pinhole camera from 3D focuses on the planet to 2D focuses in a picture. The calibration network of a camera is a n-measurement lattice which resembles this.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} \end{bmatrix}$$

Figure 2.1: Calibration Matrix [7]

2.1.2 Distortion correction

When a camera tries to map 3D world to 2D image, it has so many distortions around the edges of the image. We have used Chess Board image to correct the distortions.

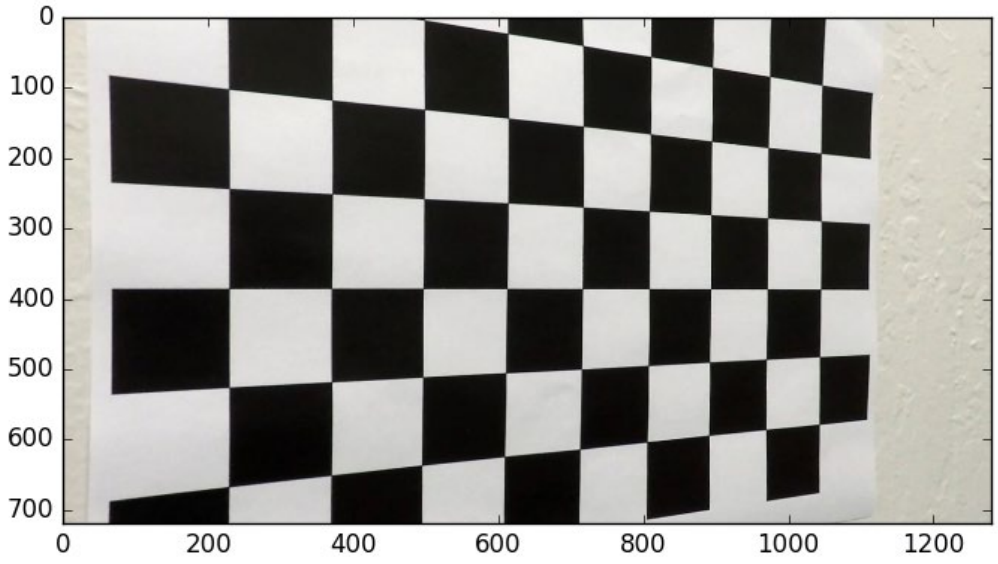


Figure 2.2: Chess Board Image [8]

Distortion correction using Chessboard Image:

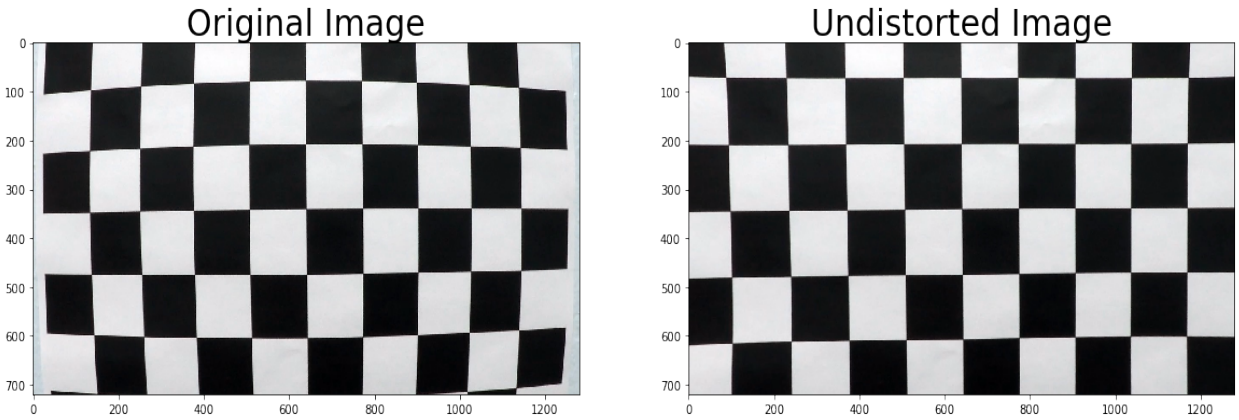


Figure 2.3: Resolution/Spatial coordinates [9]

Undistorted the Test Images:

Figure 2.4 Undistorted Test Images [10]

2.1.3 Python Code for the above mentioned Process:

```
import NumPy as np
import cv2
import glob
import pickle
Importmatplotlib.pyplotasplt
Fromosimport path



defcalibrate_camera(nx, ny, basepath):
"""

:paramnx: number of grids in x axis
:paramny: number of grids in y axis
:parambasepath: path contains the calibration images
:return: write calibration file into basepath as calibration_pickle.p
"""


objp=np.zeros((nx*ny,3), np.float32)
objp[:,:2] =np.mgrid[0:nx,0:ny].T.reshape(-1,2)


# Arrays to store object points and image points from all the images.
objpoints= [] # 3d points in real world space
imgpoints= [] # 2d points in image plane.


# Make a list of calibration images
    images =glob.glob(path.join(basepath, 'calibration*.jpg'))


# Step through the list and search for chessboard corners
forfnamein images:
img= cv2.imread(fname)
        gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

```
# Find the chessboard corners
        ret, corners = cv2.findChessboardCorners(gray, (nx,ny),None)


# If found, add object points, image points
if ret ==True:
objpoints.append(objp)
imgpoints.append(corners)


# Draw and display the corners
img= cv2.drawChessboardCorners(img, (nx,ny), corners, ret)
            cv2.imshow('input image',img)
            cv2.waitKey(500)


    cv2.destroyAllWindows()



# calibrate the camera
img_size= (img.shape[1], img.shape[0])
    ret, mtx, dist, rvecs, tvecs= cv2.calibrateCamera(objpoints, imgpoints, img_size, None, None)


# Save the camera calibration result for later use (we don't use rvecs / tvecs)
dist_pickle= {}
dist_pickle["mtx"] =mtx
dist_pickle["dist"] =dist
destnation=path.join(basepath,'calibration_pickle.p')
pickle.dump( dist_pickle, open( destnation, "wb" ) )
print("calibration data is written into: {}".format(destnation))


returnmtx, dist
```

```
defload_calibration(calib_file):
"""

:paramcalib_file:
:return: mtx and dist
"""
withopen(calib_file, 'rb') asfile:
# print('Load calibration data')
        data=pickle.load(file)
mtx= data['mtx']   # calibration matrix
dist= data['dist']   # distortion coefficients


returnmtx, dist



defundistort_image(imagepath, calib_file, visulization_flag):
""" undistort the image and visualization

:paramimagepath: image path
:paramcalib_file: includes calibration matrix and distortion coefficients
:paramvisulization_flag: flag to plot the image
:return: none
"""
mtx, dist=load_calibration(calib_file)


img= cv2.imread(imagepath)


# undistort the image
img_undist= cv2.undistort(img, mtx, dist, None, mtx)
img_undistRGB= cv2.cvtColor(img_undist, cv2.COLOR_BGR2RGB)


ifvisulization_flag:
imgRGB= cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        f, (ax1, ax2) =plt.subplots(1, 2)
        ax1.imshow(imgRGB)
        ax1.set_title('Original Image', fontsize=30)
        ax1.axis('off')
        ax2.imshow(img_undistRGB)
        ax2.set_title('Undistorted Image', fontsize=30)
        ax2.axis('off')
plt.show()


returnimg_undistRGB
```

```
if__name__=="__main__":


nx, ny=9, 6# number of grids along x and y axis in the chessboard pattern
basepath='camera_cal/'# path contain the calibration images. Add your images here


# calibrate the camera and save the calibration data
calibrate_camera(nx, ny, basepath)
```

2.2 Color Transformations

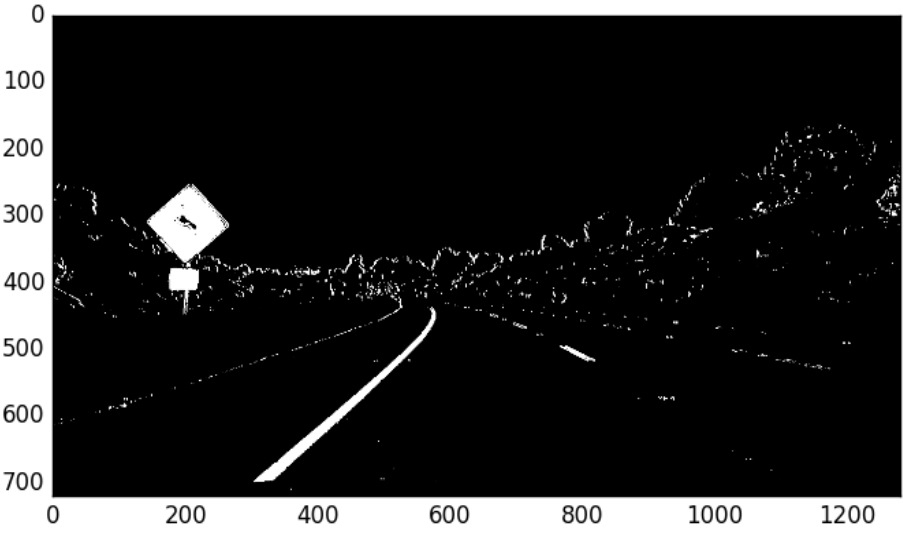
The transformation of image was done from RGB space to HLS space, and thresholding the S channel.

2.2.1 Thresholding Binary Images

The subsequent stage is to make a limit of binary picture, accepting the undistorted picture as information. The objective is to recognize pixels that are probably going to be a piece of the lane lines. Specifically, I did the following:

- Apply the accompanying channels with thresholding, to make separate "binary image" relating to every individual channel
- Absolute flat Sobel administrator on the picture
- Sobel administrator in both flat and vertical bearings and figure its size
- Sobel administrator to ascertain the heading of the angle
- Convert the picture from RGB space to HLS space, and edge the S channel
- Combine the above paired pictures to make the last binary picture

Here is the model picture (Figure 7), changed into a binary picture by joining the above threshold binary filter:



:

Figure 2.5: Threshold Binary Image [11]

2.2.2 Color Transformation Techniques used:

We used following techniques:

- Sobel Operator
- Gaussian Blur
- Canny Edge detection
- RGB to HLS

Figure 2.6: Color transformed image [12]

2.3 Perspective Transform

At the point when natural eyes see close to things, they look greater as contrast with the individuals who are far away. This is called perspective in a general manner. While transformation is the exchange of an item and so on starting with one state then onto the next.

So by doing this, the perspective transformation manages the change of 3d world into 2d picture. A similar rule on which human vision works and a similar rule on which the camera works.

2.3.1 Binary Image Rectification

Given the thresholded binary picture, the subsequent stage is to play out a perspective change. The objective is to change the picture with the end goal that we get a "superior view" of the lane, which empowers us to fit a bended line to the lane lines (for example polynomial fit). Something else this achieves is to "crop" a territory of the first picture that is well on the way to have the lane line pixels

2.3.2 Perspective Transform Techniques

To accomplish the perspective transform, we used

- OpenCV's **`getPerspectiveTransform()`**
- And **`warpPerspective()`** functions.

We hard-code the source and destination points for the perspective transform. The source and destination points were visually determined by manual inspection, although an important enhancement would be to algorithmically determine these points.

Here is the example image (Figure 9), after applying perspective transform:

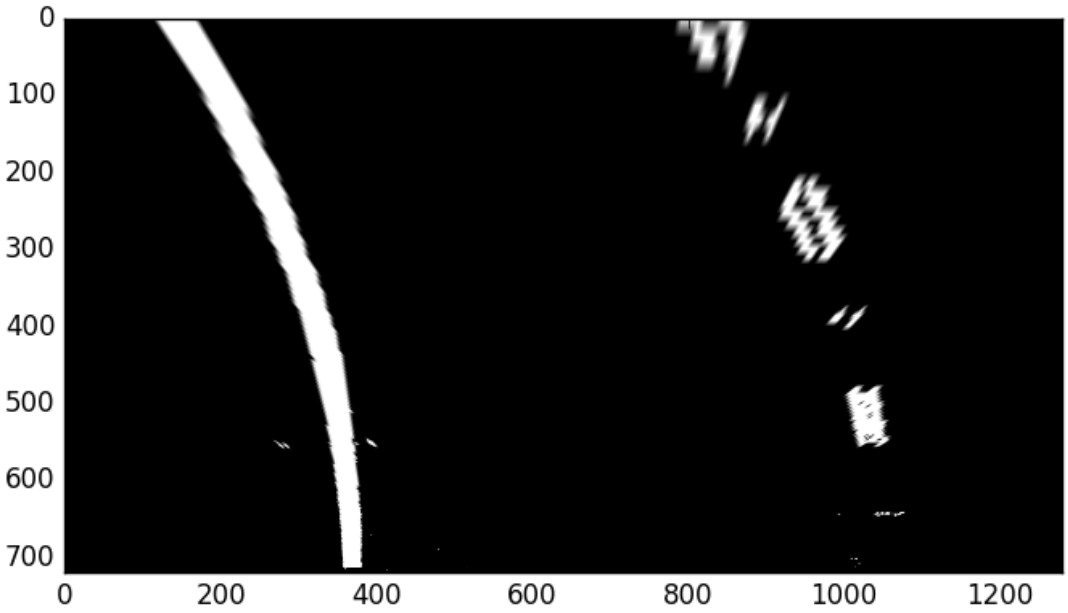


Figure 2.7: Perspective Transformed [13]

The code to perform perspective transform is in **'perspective_transform.py'**, in particular the function **perspective_transform()**. For all images in **'test_images/*.jpg'**, the warped version of that image (i.e. post-perspective-transform) is saved in **'output_images/warped_*.png'.**

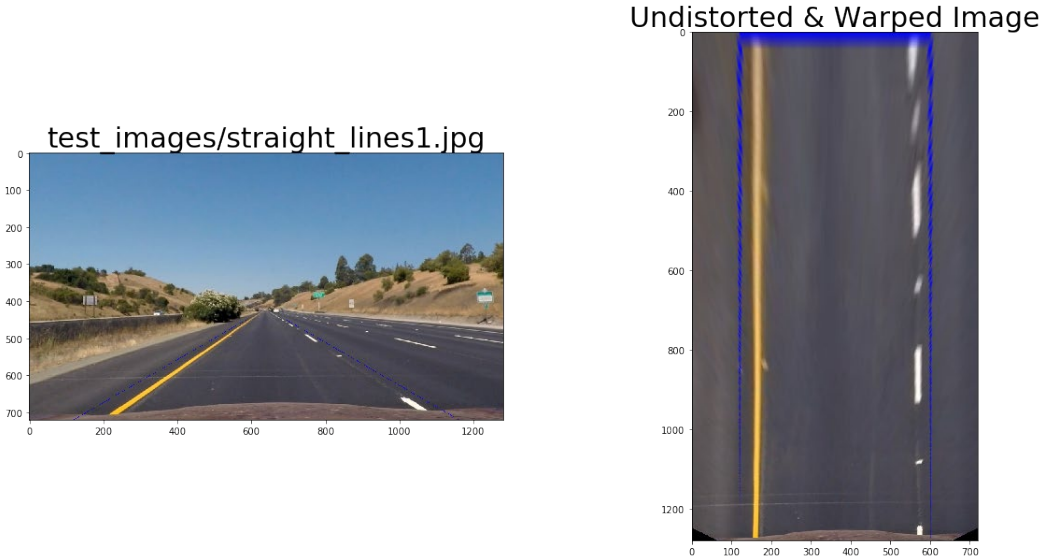


Figure 2.8: Warped and Perspective Image [14]

2.4 Lane Pixel and Curvature Detection

2.4.1 Polynomial Fitting

Given the distorted binary picture from the past advance, we presently fit a second request polynomial to both left and right lane lines. Specifically, we played out the accompanying:

•Calculate a histogram of the base portion of the picture

•Partition the picture into 9 flat cuts

•Starting from the base cut, encase a 200 pixel wide window around the left pinnacle and right pinnacle of the histogram (split the histogram down the middle vertically)

•Go up the level window cuts to see pixels that are likely as a feature of the left and right lanes, recentering the sliding windows artfully

•Given 2 gatherings of pixels (left and right lane line up-and-comer pixels), fit a second request polynomial to each gathering, which speaks to the assessed left and right lane lines

The code to play out the above is in the line_fit() capacity of 'line_fit.py'.

Since we will likely discover lane lines from a video stream, we can exploit the worldly connection between's video outlines.

Given the polynomial fit determined from the past video outline, one execution improvement I actualized is to look +/ - 100 pixels on a level plane from the recently anticipated lane lines. At that point we just play out a second request polynomial fit to those pixels found from our snappy hunt. In the event that we don't discover enough pixels, we can restore a blunder (for example return None), and the capacity's guest would overlook the present casing (for example keep the lane lines the equivalent) and make certain to play out a full hunt on the following edge. By and large, this will improve the speed of the lane indicator, valuable if we somehow happened to utilize this finder in a creation self-driving vehicle. The code to play out a curtailed search is in the tune_fit() capacity of 'line_fit.py'.

Another upgrade to misuse the fleeting relationship is to streamline the polynomial fit parameters. The advantage to doing so is make the locator progressively strong to loud info. I

utilized a basic moving normal of the polynomial coefficients (3 qualities for each lane line) for the latest 5 video outlines. The code to play out this smoothing is in the capacity add_fit() of the class Line in the record 'Line.py'. The Line class was utilized as an aide for this smoothing capacity explicitly, and Line occasions are worldwide items in 'line_fit.py'.

The following is a delineation of the yield of the polynomial fit, for our unique model picture. For all pictures in 'test_images/*.jpg', the polynomial-fit-clarified variant of that picture is spared in 'output_images/polyfit_*.png'.

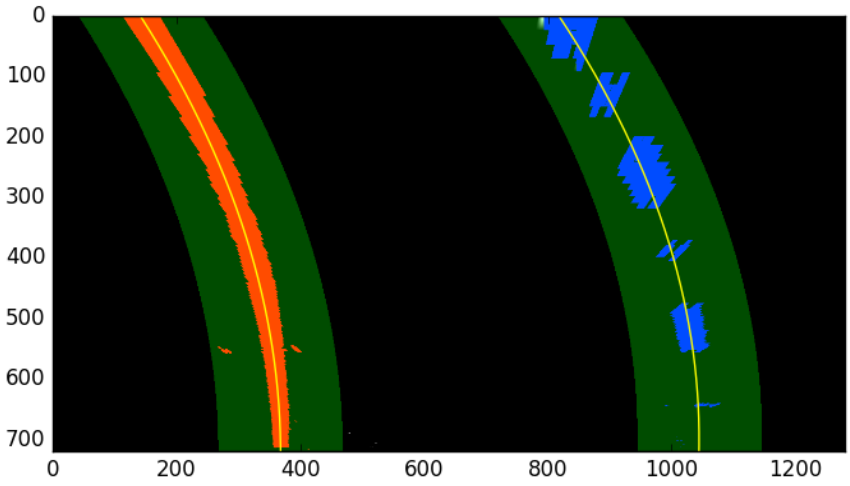


Figure 2.9: Polynomial fitting [15]

This is the most significant advance; we utilize the Hough Transform to change over the pixel specks that were recognized as edges into important lines. It takes a lot of parameters, including how straight should a line be to be viewed as a line and what ought to be the base length of the lines. It will likewise associate back to back lines for us, is we indicate the most extreme hole that is permitted. This is a key parameter for us to have the option to join a lane into a solitary distinguished lane line.

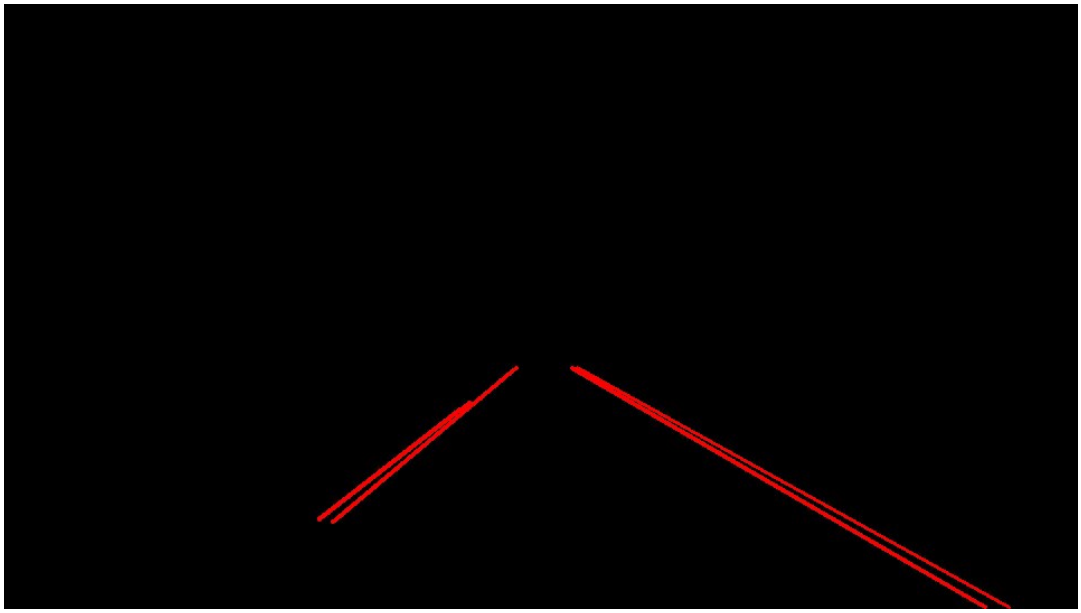


Figure 2.10 Distinguished Lane Line [16]

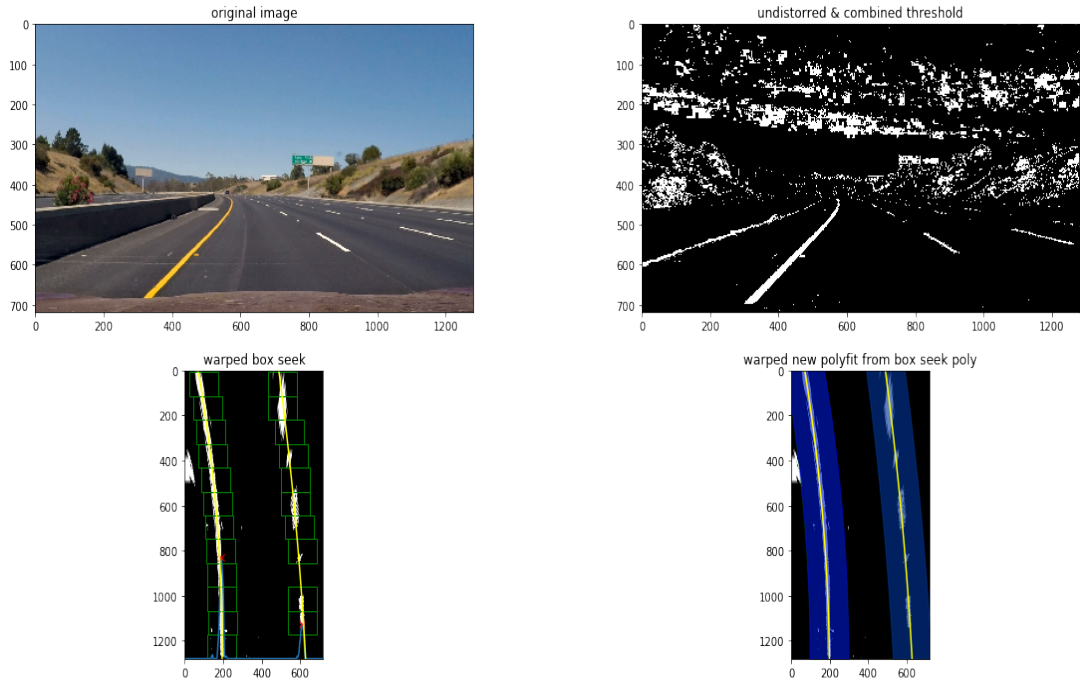


Figure 2.11 Polynomial Fitting in the original image and the undistorted threshold image [17]

2.4.2 Radius of curvature

Given the polynomial fit for the left and right lane lines, we determined the sweep of curvature for each line as indicated by recipes introduced here. We likewise changed over the separation units from pixels to meters, accepting 30 meters for each 720 pixels in the vertical heading, and 3.7 meters per 700 pixels in the flat course.

At last, we found the middle value of the span of curvature for the left and right lane lines, and detailed this incentive in the last video's explanation.

The code to ascertain the sweep of curvature is in the capacity calc_curve() in 'line_fit.py'

```
images = load_test_images()

@interact
def curvature_test(images=fixed(images), i:(0,len(images)-1)=0):
    file,img=images[i]
    binimg = binary_topdown(img)
    height = binimg.shape[0]
    histogram = lane_histogram(binimg,int(height/2),height)
    peak_left, peak_right = lane_peaks(histogram)

    # minimum distance between lanes is 3.7 meters
    lane_px_width=413
    xm_per_pix = 3.7/413 # meters per pixel in x dimension
    delta_x_pix = peak_right-peak_left
    delta_x_m = delta_x_pix * xm_per_pix
    print("left: %d, right %d - distance (%d pixel, %0.2f m) min 3.7 m" % (peak_left, peak_right,
delta_x_pix, delta_x_m))

    # dashed lane lines are 10 feet or 3 meters long each
    lane_px_height=275 # manual observation
    ym_per_pix = (3./lane_px_height) # meters per pixel in y dimension
    delta_y_pix = lane_px_height
    delta_y_m = delta_y_pix * ym_per_pix
    print("dashed lane line length (%d pixel, %0.2f m) should be 3 m" % (delta_y_pix, delta_y_m))

    boxes_left, boxes_right = calc_lane_windows(binimg)

    left_fit = calc_fit_from_boxes(boxes_left)
    right_fit = calc_fit_from_boxes(boxes_right)

    print(calc_curvature(left_fit), calc_curvature(right_fit))

    # return arr2img(binimg)
    fity = np.linspace(0, height-1, num=height)
    fitxl = poly_fitx(fity, left_fit)
    fitxr = poly_fitx(fity, right_fit)

    f, (ax1) = plt.subplots(1, 1)
    ax1.imshow(binimg,cmap='gray')
    ax1.plot(fitxl, fity, color='yellow')
    ax1.plot(fitxr, fity, color='yellow')
    ax1.set_xlim(0,720)
    ax1.set_ylim(1280,0)
```

Output:

```
left: 158, right 572 - distance (414 pixel, 3.71 m) min 3.7 m
dashed lane line length (275 pixel, 3.00 m) should be 3 m
4314.36940513 341.053223243
```

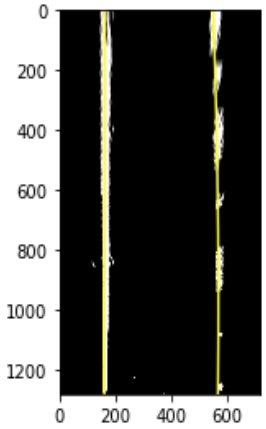


Figure 2.12 Output[18]

2.4.3 Vehicle offset from lane center

Given the polynomial fit for the left and right lane lines, I determined the vehicle's offset from the lane place. The vehicle's offset from the inside is clarified in the last video. I made indistinguishable presumptions from before while changing over from pixels to meters.

To compute the vehicle's offset from the focal point of the lane line, I expected the vehicle's inside is the focal point of the picture. I determined the lane's middle as the mean x estimation of the base x estimation of the left lane line, and base x estimation of the correct lane line. The offset is essentially the vehicle's inside x esteem (for example focus x estimation of the picture) less the lane's inside x esteem.

The code to figure the vehicle's lane offset is in the capacity calc_vehicle_offset() in 'line_fit.py'.

Figure 2.13: Vehicle offset from lane centre [19]

2.5 Lane Boundary Warping

2.5.1 Overlaying Annotation in the Original image

Given all the above mentioned, we can annotate the first picture with the lane zone, and data about the lane curvature and vehicle offset. The following are the means to do as such:

- Create a clear picture, and draw our polyfit lines (evaluated left and right lane lines)
- Fill the territory between the lines (with green shading)
- Use the opposite twist network determined from the perspective change, to "unwarp" the better than that it is lined up with the first picture's perspective
- Overlay the above explanation on the first picture

The code to play out the above is in the capacity final_viz() in 'line_fit.py'.

The following is the last annotated variant of our unique picture. For all pictures in 'test_images/*.jpg', the last annotated variant of that picture is spared in 'output_images/annotated_*.png'.

- Result with lane

:

Figure 2.14 Result [20]

2.5.2 Overlaying text in the Original image

Add text to the original image to display lane curvature and vehicle offset. We used annotate to overlay the texts on the target pixels.

The output after annotation was as follows:

Figure 2.15: Overlaying text in the Original image [21]

2.6 Testing the Approach:

In this approach we have used Open CV as core component and found the lane line. This approach used a simple hough transform line detection algorithm which serves the purpose.

Before applying the hough transform algorithm certain preprocessing steps are used. The overall process consists of the following steps:

Step 1 : Color Selection

Step 2 : Canny Edge Detection

Step 3 : Region of Interest Selection

Step 4 : Hough Transform Line Detection

2.6.1 Data pre-processing: This includes resizing and reformatting the images.

Figure 2.16: Artificial lane boundaries [22]

2.6.2 Colour selection: Selecting only yellow and white colours in the images using the RGB channels.

Figure 2.17: Colour Selected white and yellow [23]

2.6.3 RGB images are converted into HSV colour space

Figure 2.18: In HSV colour space [24]

2.6.4 RGB images are converted into HSL colour space

Figure 2.19: HSL colour space [25]

2.6.5 Filter to select those white and yellow lines

Figure 2.20: Filter for only yellow and white colour [26]

2.6.6 Canny Edge Detection: Canny edge detector for detecting edges in order to find lanes.

2.6.6.1 Gray Scaling: Converting into gray scaled in order to detect shapes in images

Figure 2.21: Gray Scaling [27]

2.6.6.2 Gaussian Smoothing (Gaussian Blur) : Making the edges smoother

Figure 2.22: Gaussian smoothing [28]

2.6.6.3 Edge Detection: Detecting edges in images

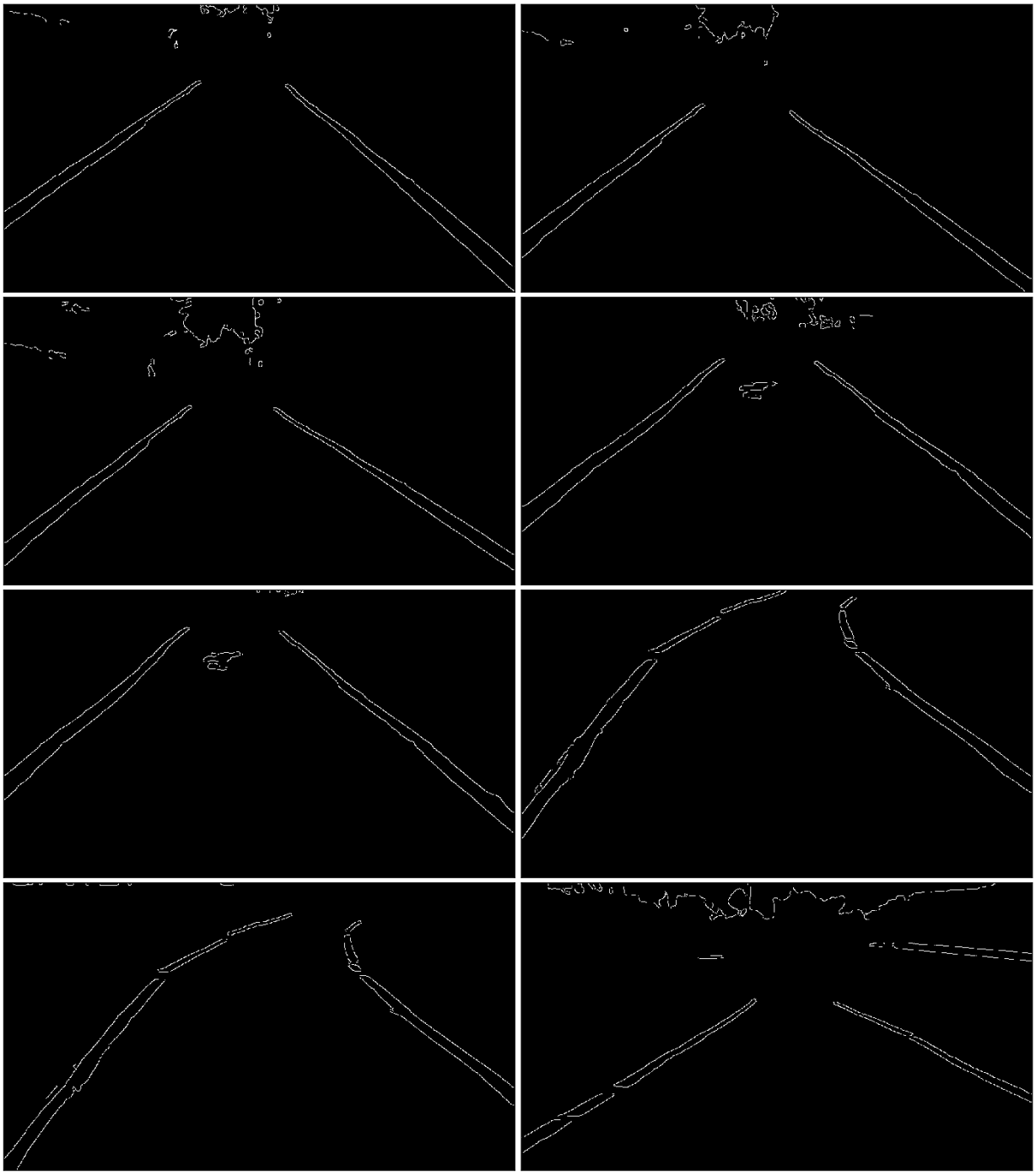


Figure 2.23: Edge Detection [29]

2.6.7 Region of Interest Selection: Selecting only region of interest

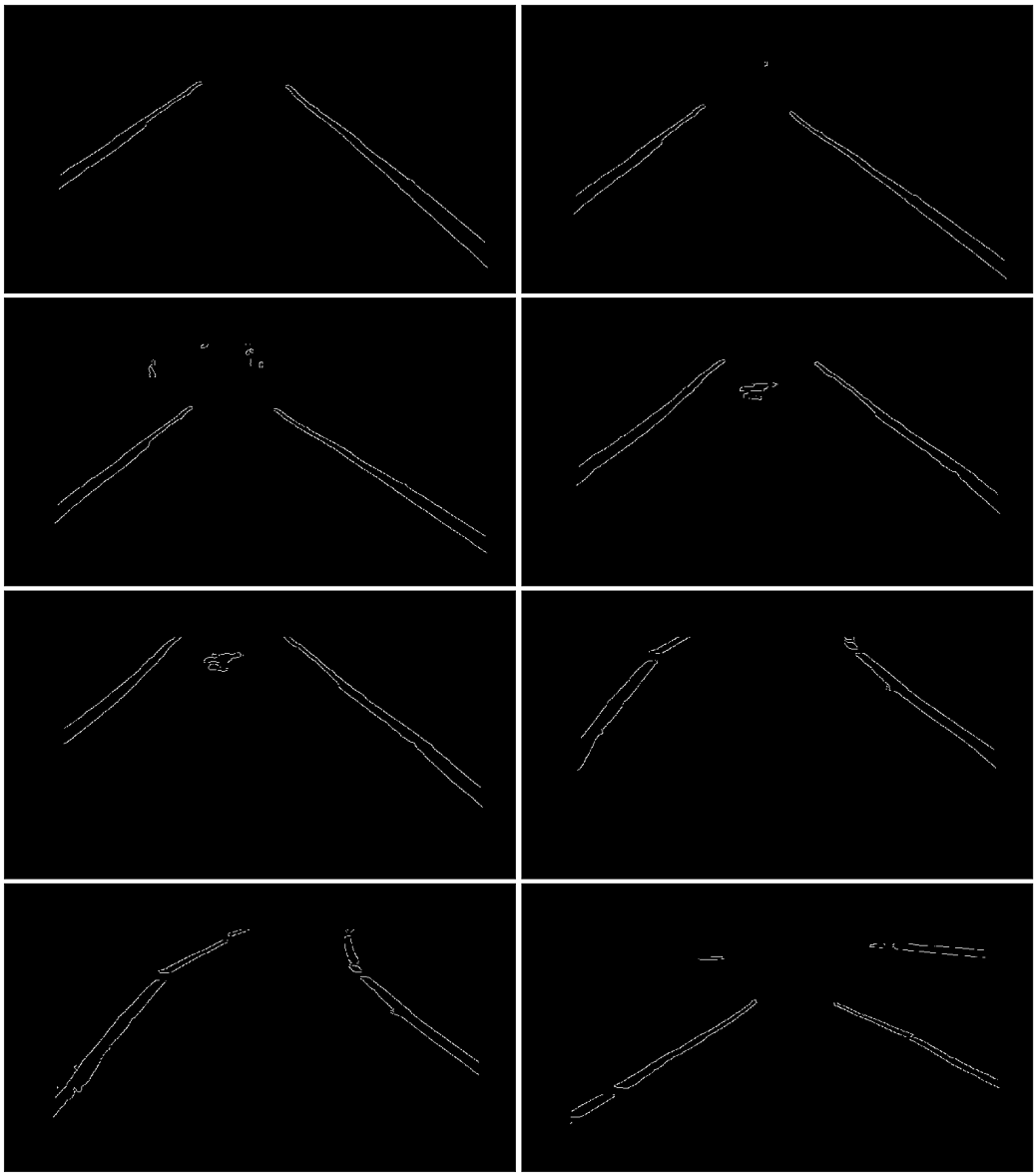


Figure 2.24: ROI Selection [30]

2.6.8 Hough Transform Line Detection: Detect lines in the edge images

Figure 2.25: Hough Transform lane detection [31]

2.6.9 Averaging and Extrapolating Lines: Averaged multiple lines detected for a lane line

Figure 2.27: Averaging Lines for representing lanes [32]

CHAPTER 3

LANE DETECTION USING CONVOLUTIONAL NEURAL NETWORK

In this approach deep learning is used to detect lanes in an image, to solve the problem of identifying curves lane which the earlier stated approach faced. As the deep learning model may be more effective at determining the important features in a given image than a human being using manual gradient and colour thresholds in typical computer vision techniques, as well as other manual programming needed within that process.

Specifically, the plan is to use a convolutional neural network (CNN), which is very effective at finding patterns within images by using filters to find specific pixel groupings that are important. The aim of this approach is for the end result to be both more effective at detecting the lines as well as faster at doing so than common computer vision techniques.

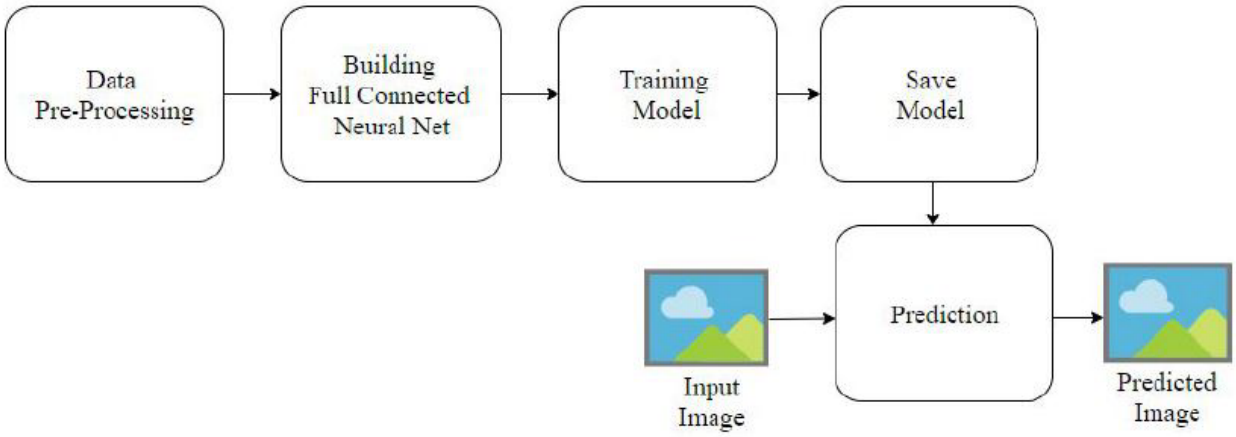


Figure 3.1: Input pipeline of this Approach [33]

The input pipeline illustrated in the above diagram is followed to solve the lane detection problem using deep learning and fully connected CNN neural nets.

3.1 What are Convolutional Neural Networks?

When we see things, what our brain is looking for are features. Depending on the features that it sees and depending on the features that it processes it categorises things in certain way depending on which features our brain picks up it will classify an image as one or the other.

In convolutional neural networks the way a computer classifies images on the basis of features is similar to the way our brain does it. We create an artificial neural network that is an artificial model of our biological neural network.

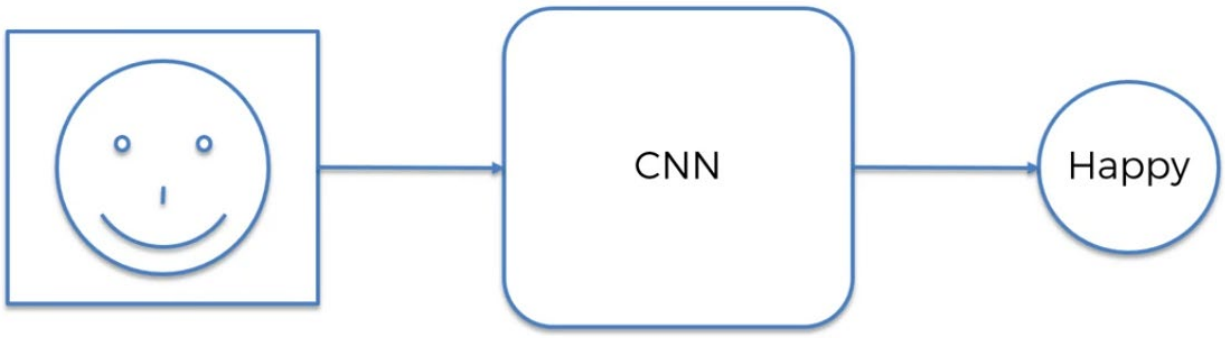


Figure 3.2: Feature detection [34]

CNN can recognise features which are very powerful in terms of many applications it has.

For Example how self driving cars recognise:

- Stop signs
- Lanes
- Vehicles
- People on the road
- Curvature
- Zebra Crossing

3.2 How does a Convolutional Network recognise features?

Neural networks leverage the fact that the black and white image is a two dimensional array.

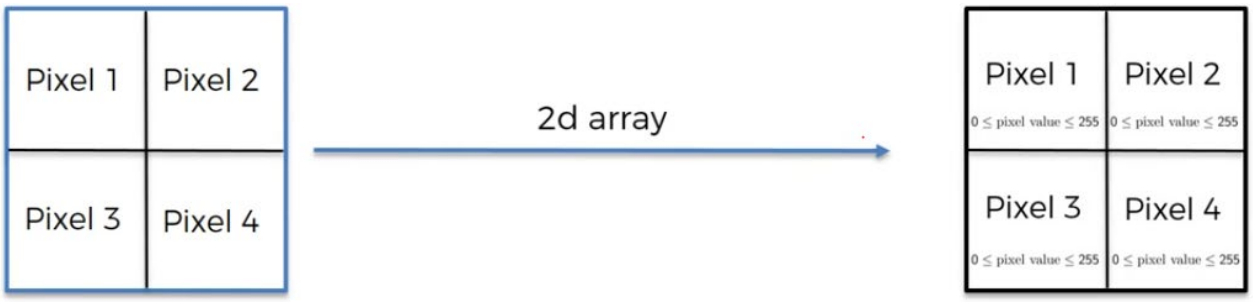


Figure 3.3: Black and white Image as 2D matrix [35]

In computer terms it's actually a two dimensional array with every single one of those pixels having a value between 0 and 255. So that's eight bits of information to the two to the power of eight is 256. So therefore the values from 0 to 255 and that's intensity of the color and in this case the color white, so 0 will be a completely black pixel and 255 will be a completely white pixel and between them you have the grayscale range of possible options for this pixel. Based on that information computers are able to then work with the image, the starting point that any image is actually has a digital representation has a digital form. Those are basically ones and zeros that form a number 0 to 255 for every single pixel.

In a coloured image it's a three dimensional array of Red, Green and Blue colours. Each colour has it’s own intensity. Every pixel has 3 values assigned to it between 0 to 255 and therefore the computer can find out what colour exactly this pixel is.

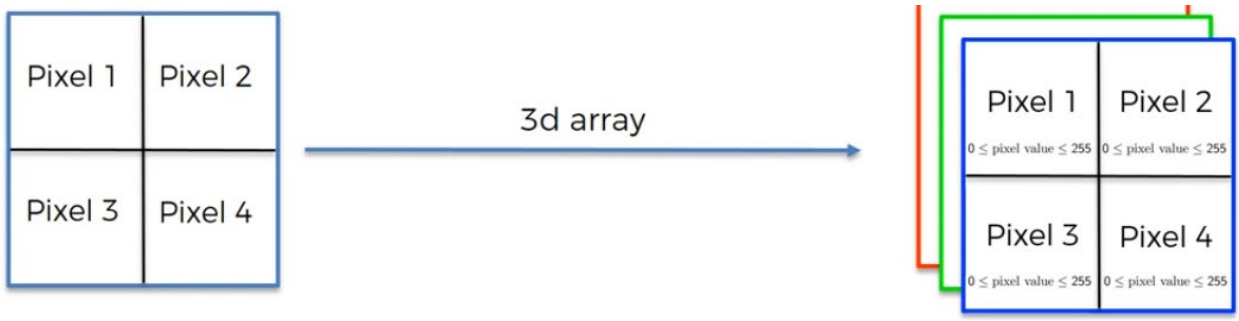


Figure 3.4: Coloured Image as a 3D matrix [36]

For example the image below can be represented as a matrix if we take white as 0 and 1 as completely black pixel.

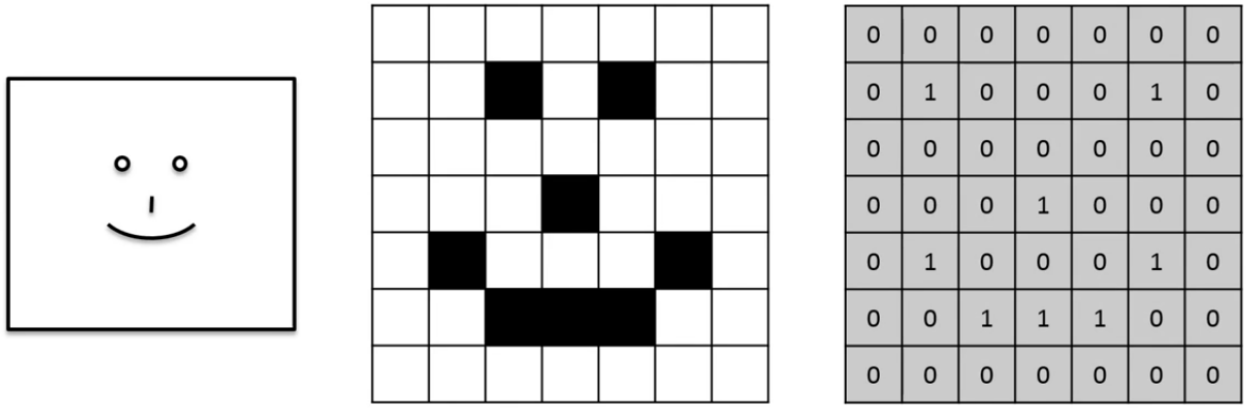


Figure 3.5: Feature as matrix with white as 0 and black as 1 [37]

3.3 Convolution operation

As we have studied before the convolution operation can be mathematically represented as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

Convolution equation

Our next task was to apply this operation on the matrix of the input image which we obtained above by representing it on the basis of the intensity of each pixel. The feature detector is also known as Filter or Kernel which is usually a 3 x 3 matrix. On a intuitive level the Feature detector is like a filter that we impose on all the nine pixels of the input image and do element wise multiplication of these matrices. The figure below signifies our result which is called as the feature map.

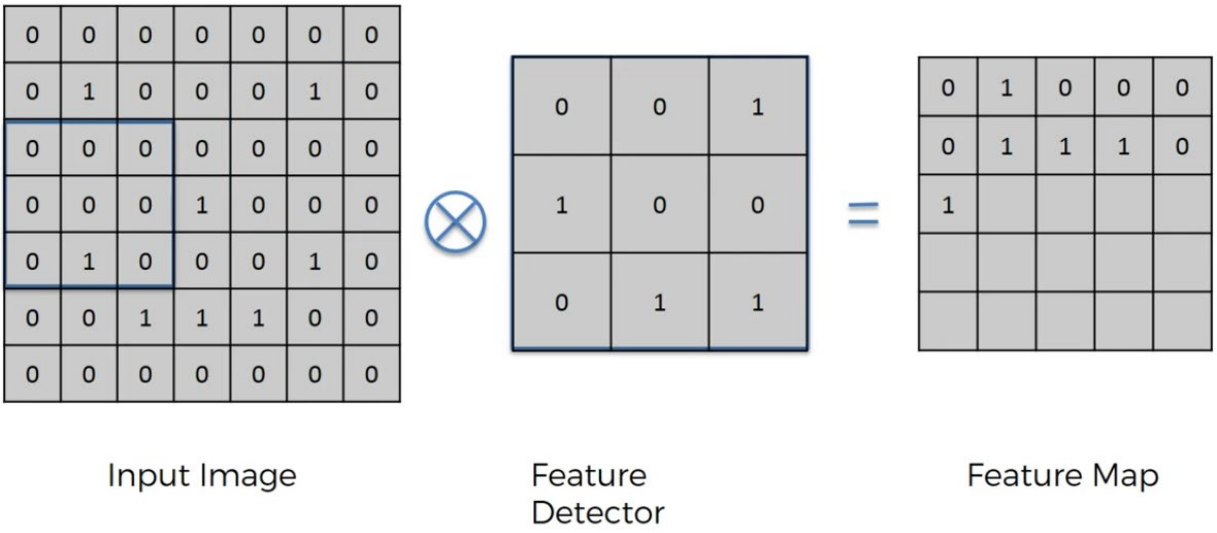


Figure 3.6: Input Image Convoluted with Feature detector [38]

The important feature of this process is that by applying the convolution operation on the feature detector reduces the size of the image. The important function of the whole convolution step is to make the image smaller because it will be easier to process it. When we have images of very large size, this step reduces the size of the image by preserving the features of the image as well. The purpose of the feature detector is to detect certain features of the image that are integral.

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Figure 3.7: Feature map [39]

For instance if we think about it this way the feature detector has a certain pattern on it. The highest number in the above feature map is when that pattern matches up.

When all the pixels matches up for example the 4 pixel in the above matrix. That is exactly where the feature is detected. We see things is how we recognise them, We don't look at every single pixel in what we see on an image or in real life we look at features.

The next step is to apply different feature maps to the input image that will detect different features. Different feature maps will also detect different features as well.

For example: The given feature detector below is used for edge enhancement:

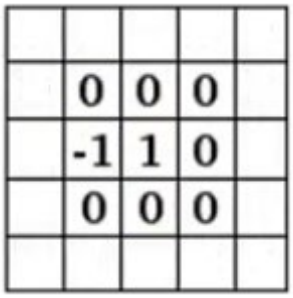


Figure 3.8: Edge enhancement Feature detector [40]

Another feature detector is used for edge detection:

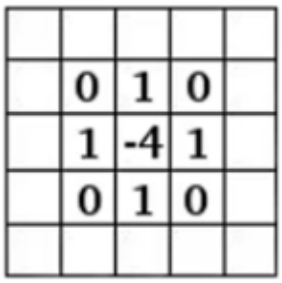


Figure 3.9: Edge Detection Feature detector [41]

The key takeaway is that the primary purpose of convolution is that we can use convolution to find features in an image using the feature detector put them into a feature map and by having them in a feature map it still preserves the spatial relationships between pixels which is very important for us. And at the same time it's important to understand that most of the time the features a neural network.

3.4 Max pooling

We have to make sure that our neural network has a property called spatial invariance meaning that it doesn't care where the features are because it doesn't have to care if the features are a bit tilted, if the features are a bit different in texture, if the features are a bit closer of features or a bit further apart relative to each other. So if the feature itself is a bit distorted our neural network has to have some level of flexibility to be able to still find that feature which is why we do Max pooling.

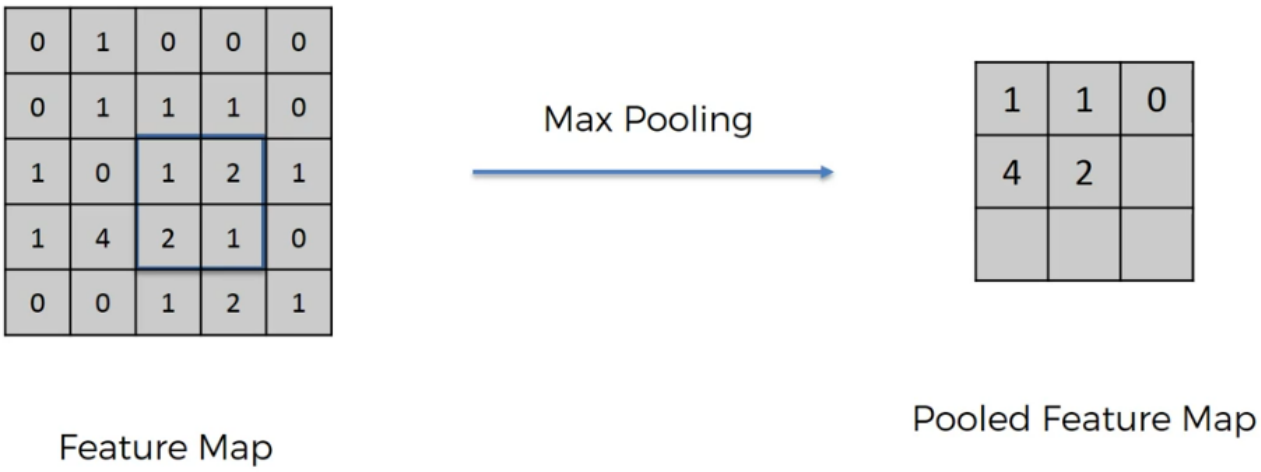


Figure 3.10: Max Pooling [42]

What we do in Max pooling is that we consider a 2 x 2 matrix from the feature map that we obtained and we take it’s maximum value. By doing this first of all we still were able to preserve the features right. We know that the maximal or the large numbers in your feature map they represent where you actually found the closest similarity to a feature. But by then pooling these features we are first of all getting rid of 75 percent of the information that is not the feature which is not the important things and then also because we are taking the maximum of the pixels, we are therefore accounting for any distortion.

This is the main feature behind Max pooling that we are still being able to preserve the features and moreover account for their possible spatial or textural or other kind of distortions. In addition to all of that we are reducing the size so there's another benefit.

That way our model won't be able to over fit onto that information because that information is not well even for human as humans it's important to see exactly the features rather than all this other noise that is coming into our eyes. Same thing for neural networks they by disregarding the unnecessary non-important formation we're helping with preventing of over fitting.

3.5 Flattening

Flattening is basically we take the pooled feature map and we flatten it into a column. We take the numbers row wise and store them in a long array column. The reason for this is because later we want to input this into an artificial neural network for further processing.

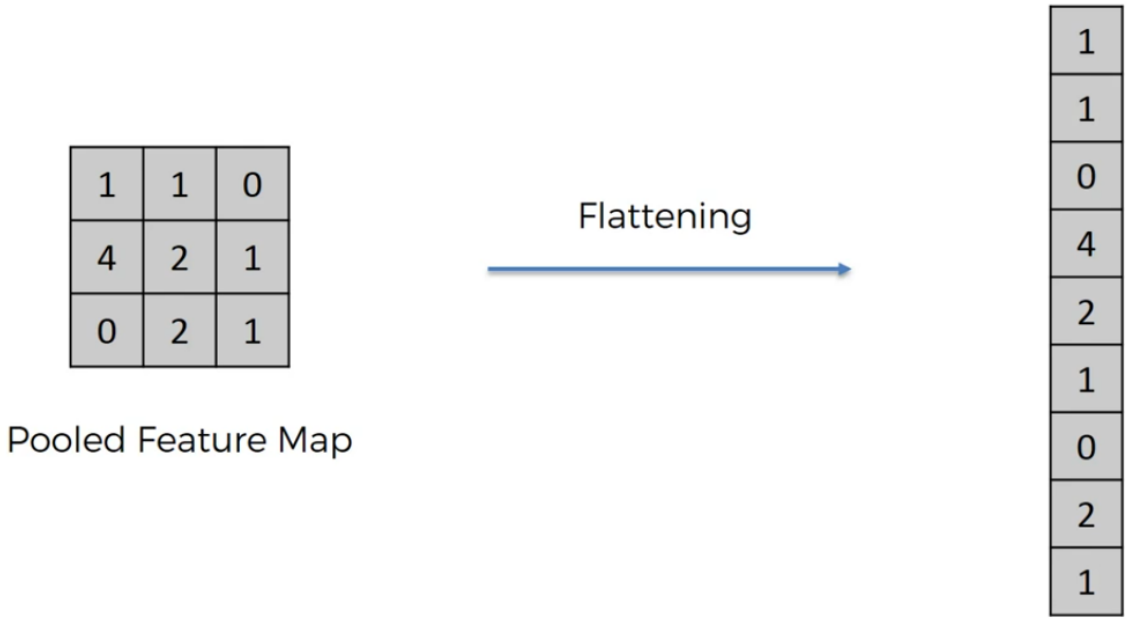


Figure 3.11: Flattening [43]

3.6 Building Fully Connected Neural Net

In this phase building of a neural network architecture is started which consist of 7 convolutional layers, 3 polling layers, 3 upsampling layer and 7 deconvolution layers, built with Keras on top of Tensorflow. The neural net assumes the inputs to be road images in the shape of 80 x 160 x 3 with the labels as 80 x 160 x 1. The output layer of deconvolution layer is a filled with the lane detected as a predicted image.

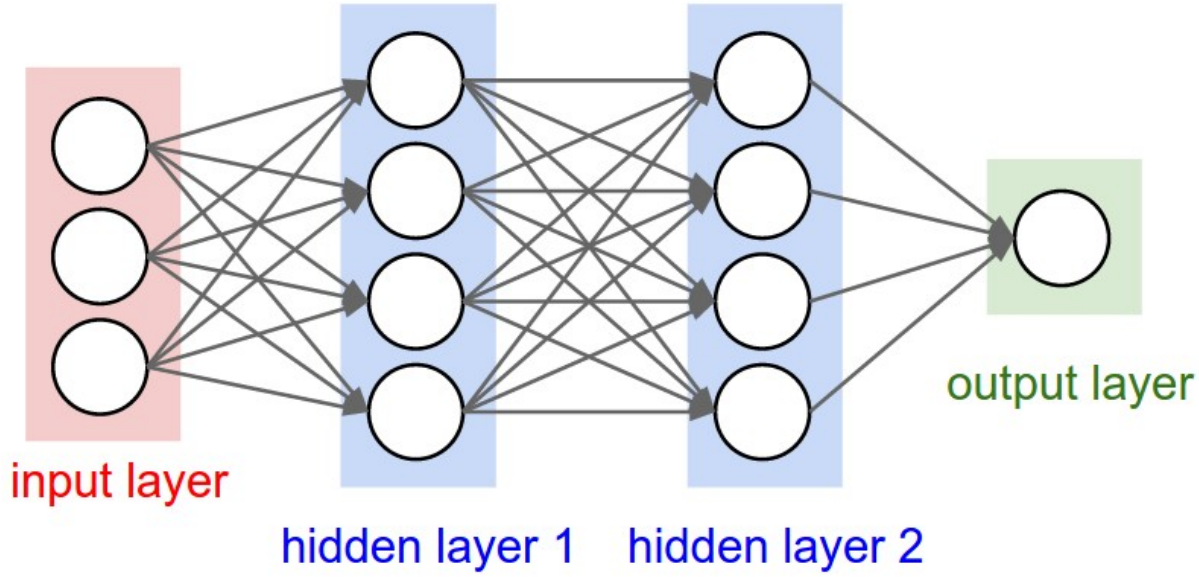


Figure 3.12: Basic model of an Artificial Neural Network [44]

We've got the input layer we've got a fully connected plan the way the fully connected layer in the artificial neural networks call them hidden layers and what the main purpose of the artificial neural network is to combine our features into more attributes that predict the classes even better. So we already have our vector of outputs in the Flattened form. The flattened result from what we've done we have some features encoded in the numbers in that vector.

We can recognise whether it's a vehicle or not a vehicle, we are inside the lane or outside the lane and so on. But at the same time we know that we have this structure called artificial neural network which is designed which has a purpose of dealing with attributes and coming

out or dealing with features and coming up with new attributes and combining attributes together to even better predict things that we're trying to predict, we pass on those values into an artificial neural network and let it even further optimize everything that we're doing.

An important thing is that why do we have two outputs. when you're doing classification you need an output Proclus except for the exception is when you have just two clusters like we have two classes here inside lane and outside lane and we could have just done one output and made it a binary output and said One is a lane and zeros a outside lane and that would have worked but at the same time if you have more than two categories for instance cars, trucks and vans then you have to have a neuron per every category and that's why we're going to practice with two categories in this example so that we know what to expect if we ever have more than two categories. We've already done all the groundwork we've done the convolution we've done the pooling and the flattening and now the information is going to go through the artificial neural network.

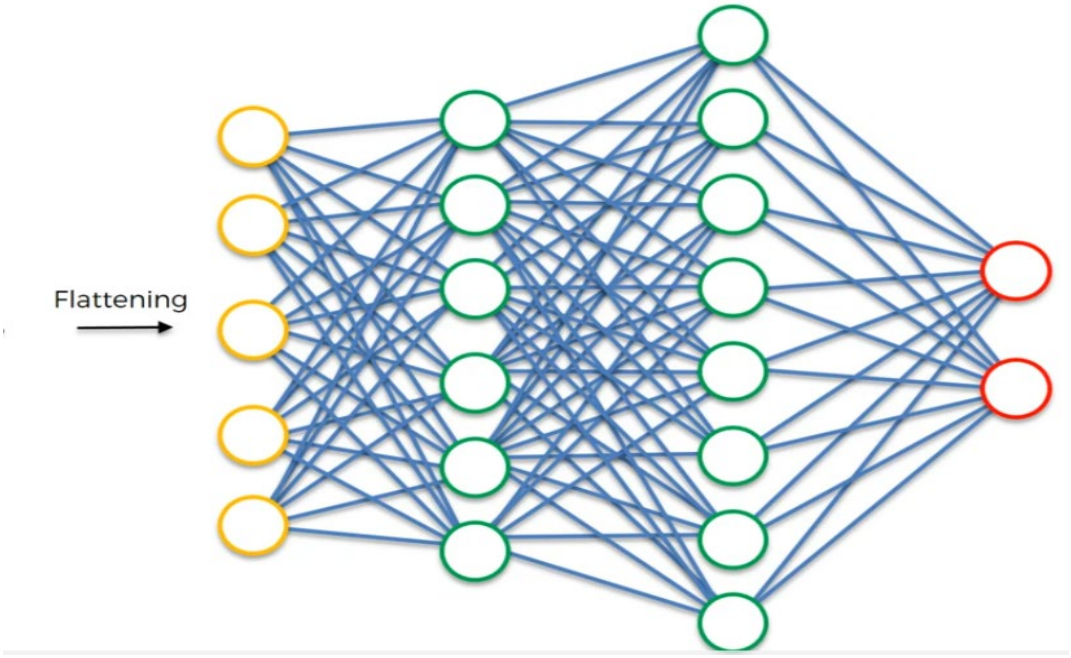


Figure 3.13: Model of a Deep Neural Network connection with weights to it [45]

There is information going through from the very start from the moment when the image is processed and convolved then pooled, flattened and then through the artificial neural network all four steps and then a prediction is made let's just say a prediction is made.

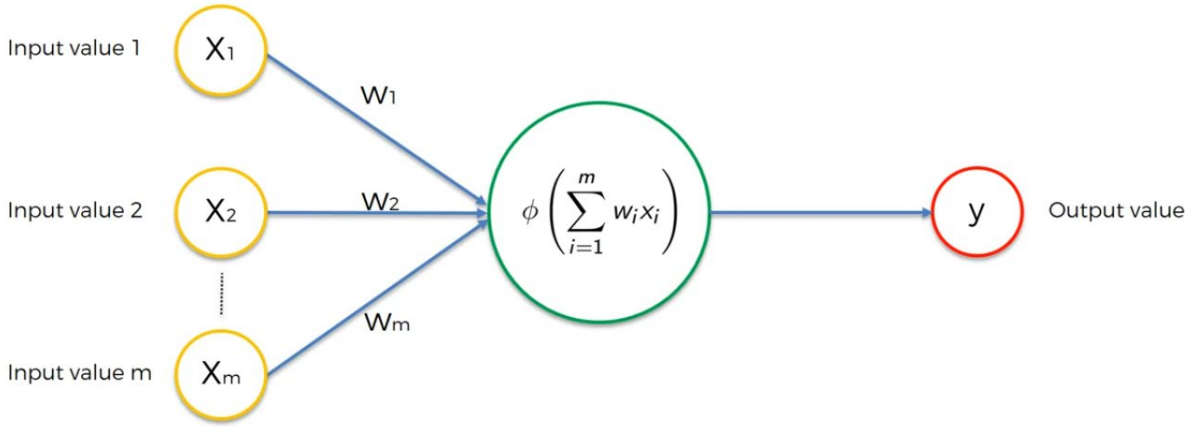


Figure 3.14: A neuron in the neural network [46]

Each neuron in a neural network has inputs and outputs to it with weights assigned to it, when a prediction is made and the predicted results are correct the neurons send a positive feedback to the network and if it turns out to be wrong and then an error is calculated. It's called a loss function and we use a cross entropy function for that. The weights are readjusted and the feature map is changed to get better results.

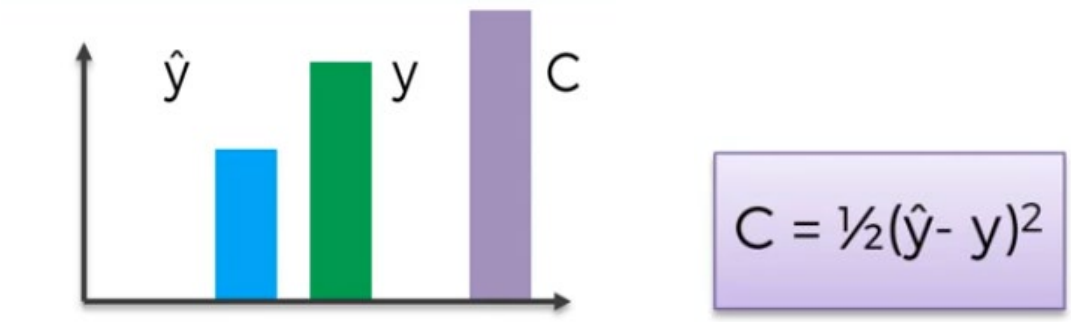


Figure 3.15: Cost Function and Error calculation [47]

But we have lots of type of function which tells us how well our network is performing and we're trying to optimize it or minimize that function to optimize our network. So the error is calculated and then it's back propagated through the network just like we had in artificial neural networks is back propagated and the some things are adjusted in the network to help optimize the performance and the things that are adjusted are as usual the weights in the

artificial neural network Then also another thing that is adjusted is the feature detectors so we know that we're looking for features but what if we're looking for the wrong features. They are adjusted so that maybe next time it'll be better. The feature detectors are adjusted the weights are adjusted and this whole process happens again and then again the errors back propagate.

This keeps going on and on and that's how our network is optimized that's how our network trains on the data.

So the important thing here is that the data goes through the whole area from the very start to the very end then the error is compared so the error is calculated and then is back propagated. So with artificial neural networks this is how we classify objects.

3.6.1 Source code of the model:

```
import numpy as np
import pickle
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

# Import necessary items from Keras
from keras.models import Sequential
from keras.layers import Activation, Dropout, UpSampling2D
from keras.layers import Deconvolution2D, Convolution2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.preprocessing.image import ImageDataGenerator

# Load training images
train_images = pickle.load(open("train_images.p", "rb" ))

# Load image labels
labels = pickle.load(open("train_labels.p", "rb" ))

# Make into arrays as the neural network wants these
train_images = np.array(train_images)
labels = np.array(labels)

# Normalize labels - training images get normalized to start in the network
labels = labels / 255

# Shuffle images along with their labels, then split into training/validation sets
train_images, labels = shuffle(train_images, labels)

X_train, X_val, y_train, y_val = train_test_split(train_images, labels, test_size=0.1)

# Batch size, epochs and pool size below are all paramaters to fiddle with for optimization
batch_size = 150
epochs = 20
pool_size = (2, 2)
```

```
input_shape = X_train.shape[1:]

model = Sequential()
# Normalizes incoming inputs. First layer needs the input shape to work
model.add(BatchNormalization(input_shape=input_shape))

# Conv Layer 1
model.add(Convolution2D(60, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu', name = 'Conv1'))

# Conv Layer 2
model.add(Convolution2D(50, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu', name = 'Conv2'))

# Pooling 1
model.add(MaxPooling2D(pool_size=pool_size))

# Conv Layer 3
model.add(Convolution2D(40, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu', name = 'Conv3'))
model.add(Dropout(0.2))

# Conv Layer 4
model.add(Convolution2D(30, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu', name = 'Conv4'))
model.add(Dropout(0.2))

# Conv Layer 5
model.add(Convolution2D(20, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu', name = 'Conv5'))
model.add(Dropout(0.2))

# Pooling 2
model.add(MaxPooling2D(pool_size=pool_size))

# Conv Layer 6
model.add(Convolution2D(10, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu', name = 'Conv6'))
model.add(Dropout(0.2))
```

```
# Conv Layer 7
model.add(Convolution2D(5, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu', name = 'Conv7'))
model.add(Dropout(0.2))

# Pooling 3
model.add(MaxPooling2D(pool_size=pool_size))

# Upsample 1
model.add(UpSampling2D(size=pool_size))

# Deconv 1
model.add(Deconvolution2D(10, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu',
                          output_shape = model.layers[8].output_shape, name = 'Deconv1'))
model.add(Dropout(0.2))

# Deconv 2
model.add(Deconvolution2D(20, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu',
                          output_shape = model.layers[7].output_shape, name = 'Deconv2'))
model.add(Dropout(0.2))

# Upsample 2
model.add(UpSampling2D(size=pool_size))

# Deconv 3
model.add(Deconvolution2D(30, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu',
                          output_shape = model.layers[5].output_shape, name = 'Deconv3'))
model.add(Dropout(0.2))

# Deconv 4
model.add(Deconvolution2D(40, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu',
                          output_shape = model.layers[4].output_shape, name = 'Deconv4'))
model.add(Dropout(0.2))
```

```
# Deconv 5
model.add(Deconvolution2D(50, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu',
                          output_shape = model.layers[3].output_shape, name = 'Deconv5'))
model.add(Dropout(0.2))

# Upsample 3
model.add(UpSampling2D(size=pool_size))

# Deconv 6
model.add(Deconvolution2D(60, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu',
                          output_shape = model.layers[1].output_shape, name = 'Deconv6'))

# Final layer - only including one channel so 1 filter
model.add(Deconvolution2D(1, 3, 3, border_mode='valid', subsample=(1,1), activation = 'relu',
                          output_shape = model.layers[0].output_shape, name = 'Final'))


datagen = ImageDataGenerator()
datagen.fit(X_train)

# Compiling and training the model
model.compile(optimizer='Adam', loss='mean_squared_error')
model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size), samples_per_epoch = len(X_train),
                    nb_epoch=epochs, verbose=1, validation_data=(X_val, y_val))

# Save model architecture and weights
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)

model.save_weights('model.h5')

# Show summary of model
model.summary()
```

3.7 Dataset

The dataset used for this approach is a BDD100K Lane Marking Dataset. The dataset consist of Images and label. In this phase the dataset is read and then saved as a pickle files with “train_images.p” and “train_labels.p”. This is the data that was initially used to train the model, after that we used the dataset which was collected by us for training the model with practical and familiar surroundings. The model was trained and tested with this data set

Figure 3.16: Dataset collected for straight, curved roads and slopes [48]

3.8 Training Model

Before the model training start the dataset is divided into training set and a validation set created using sklearn train_test_split method. The model is compiled with mean squared error as the loss function and trained over 20 epoch on the whole training dataset and validated on the validation dataset.

3.9 Save Model

After the model training is completed. This trained model is saved in a json along with its weights. Now this saved model can be used for doing prediction on the unknown data.

3.10 Prediction

In this phase the saved model is loaded along with its weights. The test image is pre processed to be of the shape 80 x 160 x 3, as the model expect that as the input shape. Once done the pre processed image is given as an input to the models which predicts the output over which the filling of green colour is added as an region of the lane detected.

3.11 Performances Metrics

The performances metrics to be used for evaluating the build model are confusion matrix, Precision, recall, ROC curve, PR curve, Mean Squared Error(MSE) and mAP score.

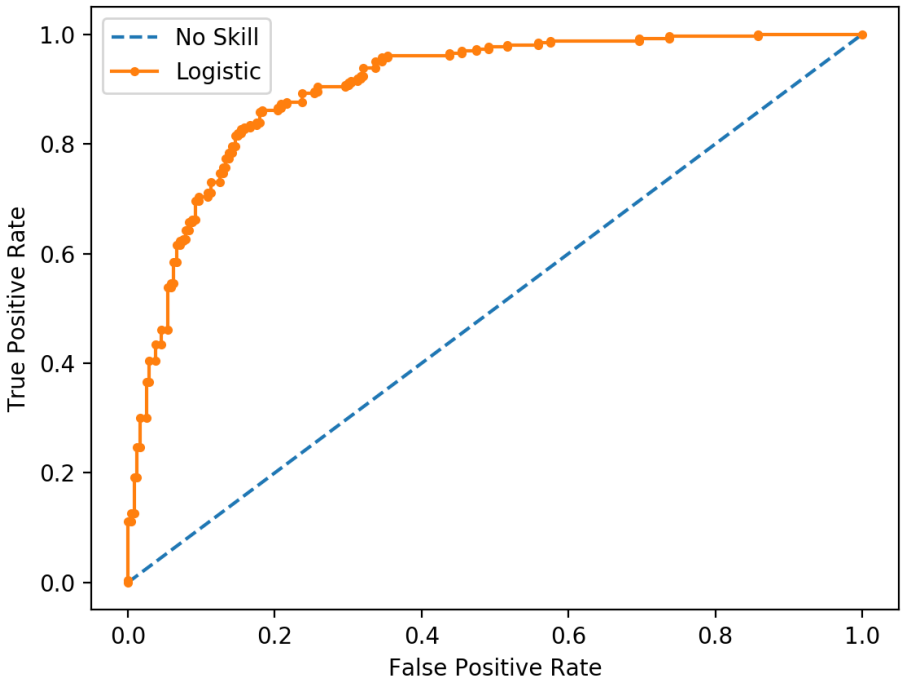


Figure 3.17: ROC curve to determine Accuracy of the Result [49]

3.12 Testing the model:

In [1]:
```
# import the libaries

import numpy as np
import cv2,glob,os
import matplotlib.image as mpimg
from scipy.misc import imresize
from moviepy.editor import VideoFileClip
from IPython.display import HTML
from keras.models import model_from_json
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

In [2]:
```
# plot the testing image

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

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

show_images(test_images)
```

In this part we are importing libraries to use further in our code. The next step is to resize the Test images that we are going to be feeding into our model. To see how our image will be going into the model we are also plotting them using pyplot function in the matplotlib library.

Figure 3.18: Plotted the Test set [50]

In [3]:
```
# Load pretrained model

# load the model json file
json_file = open('model.json', 'r')
json_model = json_file.read()
json_file.close()

# load the model from the json
model = model_from_json(json_model)
# load the wieghts of the model
model.load_weights('model.h5')
```

Our next task is to load our pre trained model that we already trained before. We will Feed these test images to our model to try to predict the lanes based on their features. We have created model as an object that will read the loaded model.

In [4]:
```
# create a class Lane

# Class to average lanes with
class Lanes():
    def __init__(self):
        self.recent_fit = []
        self.avg_fit = []
```

In [5]:
```
# resize the image and predict the lane to be drawn from the model in G color

def road_lines_image(image):

    img_arr = cv2.imread(image)
    actual_image = imresize(img_arr, (720, 1280, 3))

    # Get image ready for feeding into model
    img = mpimg.imread(image)
    small_img_2 = imresize(img, (80, 160, 3))
    small_img_1= np.array(small_img_2)
    small_img = small_img_1[None, :, :, :]

    # Make prediction with neural network (un-normalize value by multiplying by 255)
    prediction = model.predict(small_img)[0] * 255

    # Add lane prediction to list for averaging
    lanes.recent_fit.append(prediction)
    # Only using last five for average
    if len(lanes.recent_fit) > 5:
        lanes.recent_fit = lanes.recent_fit[1:]

    # Calculate average detection
    lanes.avg_fit = np.mean(np.array([i for i in lanes.recent_fit]), axis = 0)

    # Generate fake R & B color dimensions, stack with G
    blanks = np.zeros_like(lanes.avg_fit).astype(np.uint8)
    lane_drawn = np.dstack((blanks, lanes.avg_fit, blanks))

    # Re-size to match the original image
    lane_image = imresize(lane_drawn, (720, 1280, 3))
```

The next step was to feed the images to the model and try to predict the lanes using our model so we created a function which will take images as input and return the images with lanes detected in green colour.

In [6]:
```
# predict the lane detector

#create a lanes object
lanes = Lanes()

for path in glob.glob('test_images/*.jpg'):
    res_img = road_lines_image(path)
    names = [os.path.basename(x) for x in glob.glob(path)]
    out_path = 'test_predict/'+names[0]
    # save the result in a image
    cv2.imwrite(out_path,res_img)
```

In [7]:
```
# plot the predicted result

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

show_images(predicted_images)
```

Then we created a loop that will input all the Test Images from that particular folder and call our function to predict these images. Now, the last task was to display the images that returned from the function.

Figure 3.19: Predicted Result [51]

3.13 Observations and Conclusion:

As we can observe above that the artificial neural network seem to detect lanes itself without any human input, It is easily detecting lanes on straight roads and turns but it is having problems detecting lanes at slopes downhill and uphill. To improve this we can train our model using more dataset of hilly region and we can also increase the depth of our network by adding more layers to it. So far the accuracy of our model has been up to 82.34% but it can be increased by taking the above mentioned measures. The important thing to note is that by using machine learning we were able to increase the efficiency of lane detection significantly and the artificial neural network is more flexible in terms of input that it takes and this has more variety of applications to offer. At a much complex level we can also do other operations like real time vehicle detection and traffic signs detection with this system based on video input however we were able to get correct results in our project with 82% accuracy. Once we can increase this accuracy to 100% and create a system that we can rely on we will see self driving cars roll out on roads. Companies like Mercedes and Google are constantly researching and developing techniques to perfect this project because it has so much to offer and it is safe to say that once this idea is perfected it will be a game-changing idea in the automobile industry.

REFERENCES

[1] Johann Borenstein & Yoram Koren, *Obstacle Avoidance with Ultrasonic Sensors*, IEEE *JOURNAL OF ROBOTICS AND AUTOMATION*, vol. 4, no 2, APRIL I988, pp. 213-218.

[2] Yue Wanga, Eam Khwang Teoha & DinggangShenb, *Lane detection and tracking using B-Snake, Image and Vision Computing* 22 (2004) , available at:www.elseviercomputerscience.com, pp. 269–280.

[3] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski. *Self-supervised monocular road detection in desert terrain.* G. Sukhatme, S. Schaal, W. Burgard, and D. Fox, editors& Proceedings of the Robotics Science and Systems Conference, Philadelphia, PA, 2006.

[4] Joel C. McCall &Mohan M. Trivedi, *Video-Based Lane Estimation and Tracking for Driver Assistance: Survey, System, and Evaluation, IEEE Transactions on Intelligent Transportation Systems,* vol. 7, no 1, March 2006, pp. 20-37.

[5] Tushar Wankhade& Pranav Shriwas, *Design of Lane Detecting and Following Autonomous Robot, IOSR Journal of Computer Engineering* (IOSRJCE) ISSN: 2278-0661 Volume 2, Issue 2 (July-Aug. 2012), pp. 45- 48.

[6] Xiaodong Miao, Shunming Li & Huan Shen*, On-Board lane detection system for intelligent vehicle based on monocular vision, International Journal on Smart Sensing and Intelligent Systems*, vol. 5, no 4, December 2012, pp. 957-972.

[7] A. Bar Hillel, R. Lerner, D. Levi, & G. Raz. *Recent progress in road and lane detection: a survey. Machine Vision and Applications,* Feb. 2012, pp. 727–745.

[8] Li, M., Zhao, C., Hou, Y. & Ren, M. , *A New Lane Line Segmentation and Detection Method based on Inverse Perspective Mapping, International Journal of Digital Content Technology and its Applications.* vol. 5, no 4, April 2011, pp. 230-236.

[9] Girish Varma, Anbumani Subramanian, Anoop Namboodiri, Manmohan Chandraker & C V Jawahar - IDD: *A Dataset for Exploring Problems of Autonomous Navigation in Unconstrained Environments* - IEEE Winter Conf. on Applications of Computer Vision (WACV 2019).

[10] Chuan-en Lin: Tutorial: *Build a lane detector*, 2018 Dec. 17. Available: https://towardsdatascience.com/tutorial-build-a-lane-detector-679fd8953132.[Apr 06, 2019]

[11] Aydin Ayanzadeh: "*Udacity Advance Lane-Detection of the Road in Autonomous Driving"* 2018 , Mar. 19. Available:https://medium.com/deepvision/udacity-advance-lane-detection-of-the-road-in-autonomous-driving-5faa44ded487.[Apr 06, 2019]

[12] Udacity: *self-driving-car*, (2017), GitHub repository.https://github.com/udacity/self-driving-car.[Apr 06, 2019]

[13] Kirill Eremenko, Hadelin De Ponteves, SuperDataScienceTeam Udemy *A-Z: Machine Learning with python* https://www.udemy.com/course/machinelearning/ [May 2020]