Machine Learning Engineer Nanodegree

Capstone Project

Abhishek Kumar Dubey

August 14th, 2018

I. Definition

Project Overview

The project is intended to assist the driver while driving. If someone is there who can guide us while driving so that we do not get off lane it will be a boon for us. Now a days a lot of accident occurs just because of the ignorance of the driver. But if the driver is keep informed if is off lane the chances of the accident would decrease drastically.

We cannot appoint an extra human beside the driver to take care of him and make him alert if makes a mistake. Even though we appoint a human assistance for the driver there is no guarantee the assistance will not commit accident. Road accidents are one of the biggest cause of deaths on Indian roads. As per media statistics, in India one person is killed in a road accident every four minutes. The causes for India's exceptionally high number of on road casualties include - bad road, careless user behavior, and defective road design and engineering, poor enforcement of traffic rules and the lack of rapid trauma care.

According to one report:

One serious road accident in the country occurs every minute and 16 die on Indian roads every hour.

1214 road crashes occur every day in India.

20 children under the age of 14 die every day due to road crashes in in the country.

Two people die every hour in Uttar Pradesh – State with maximum number of road crash deaths.

Tamil Nadu is the state with the maximum number of road crash injuries

All the problems can be solved if we can appoint an assistance to the driver which never makes the mistake or the assistance takes over the control of the vehicle and let the driver relax.

This the beginning of the new era – Autonomous driving. The era where no road accident will take place. This is my small contribution to bring the dream to the reality.

Automatic lane detection is the first step towards the goal. Automatic lane detection can warn the driver if he goes off lane. It will reduce the chances of the road accident drastically. If the driver is always driving in their respective lane the chances or the road accident can be decreased by the 60 percent – NDTV report says.

The problem is addressed using the neural network. The neural network works in the same way as our mind works and best part of the neural network is we can improve the efficiency beyond the capabilities of the human being.

But the neural network needs to be trained. The network must be taught to work as we teach a child to do any specific task. We can train the neural network by showing it the example. Let the network take an action and then tell the network if the action taken was correct or not. This is the way to train the network. So we need the data to train the network.

The big problem is getting the data and then labeling the data. Labeling is nothing but the correct action which we expect from the network. At the time of training the network compares with the label and comes to know if the action taken by the network was correct or not. So be need to label the data.

The data should be the combination of the all the situation which can occur in the real scenario. The data I found online but it was not labeled for the lanes. So I labeled the data using old image processing algorithm. The algorithm is similar to Udacity advance lane detection.

At first the image is converted into grayscale and then canny edge detection algorithm is applied to detect the boundary. Then bird eye view is obtain. The image is converted in bird eye view so the histogram calculation can give the estimation of the lane position. The sliding window and moving average concept is applied to calculate the lane position.

After this approach there were so many images whose labels was not correct. The Matlab ground truth encoder was used to correct them, at last manual work was done to correct some of the labels.

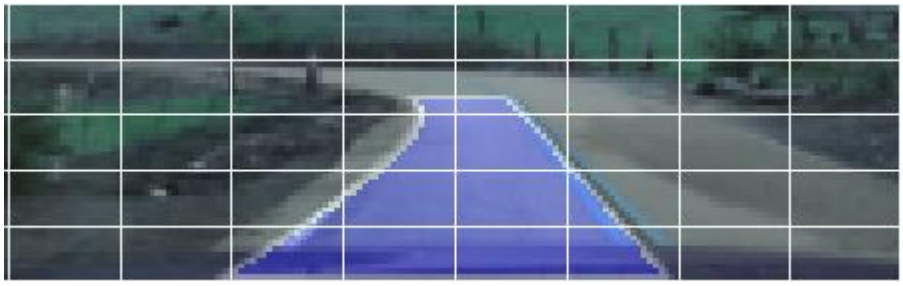
My previous work on the lane detection based on image processing (For raspberry pi) can be seen here.

Problem Statement

The problem statement is to make the computer more efficient to detect the lanes than the human as the human beings are porn to make the accident. Computers are far better than the human.

An efficient neural network will be devolved so that the problem can be solved. The problem can be better attacked using the convolutional neural network-CN N. CNN are best for image related AI. The best part of the CNN is that it sees the visual things in the same way as we do. It first classify the simple things like lines and circle in the same way as human vision cortex does. It then learns more complex patterns.

The CNN will be able to correctly classify the lanes. It will overlay the color between the two lanes. The output of the CNN will look like below



Metrics

In statistics, the mean squared error (MSE) of an estimator measures the average of the squares of the errors—that is, the average squared difference between the estimated values and what is estimated. MSE is a risk function, corresponding to the expected value of the squared error loss.

The MSE is a measure of the quality of an estimator—it is always non-negative, and values closer to zero are better.

The formula:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

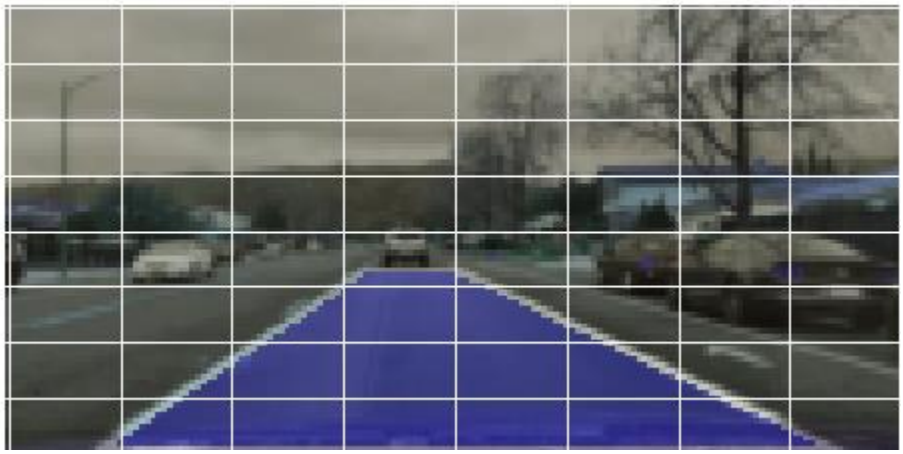
As the output of the CN N is pixels. The MSE can be better used as the performance calculation.

II. Analysis

Data Exploration

As discussed above I searched online resource for the data set. But I did not find any properly labeled data. There are many data sets available online it has labels of pedestrian, road signs, vehicle etc. But no labels for lane marking. There are some data set available with lane marking also but not with all the conditions

I took online dataset and decided to label the data myself. I used state of art method to label the data. Some of them was not labeled correctly. I used Matlab ground truth labeler to label them. At last some of the data cannot be labeled by any algorithm. I labeled those data manually. After the labelling the sample looks as below iamge:



I used my regression logic to label the data. I had followed Udacity Self driving nanodegree program in which I had implemented advanced lane detection function. I did the same to label the data automatically. After Canny edge detection I applied bird eye view and then computed histogram. In the next step I applied the moving average and window technique to filter the pixels which are good candidates for lanes. I applied the regression on those pixels to compute a 2-D polynomial equation. It was done for left and right lane.

The polynomial equation was not the last step. The polynomial was calculated for both lanes marker left and right, The area between the left and right lanes were filed with the particular color and I got final labeled dataset.

The dataset contains all type of conditions. The two player for the dataset is road curvature and the weather. The dataset statics as below:

Curvature:

1. Straight:30%
2. Moderate:30%
3. Very curvy:40%

Weather:

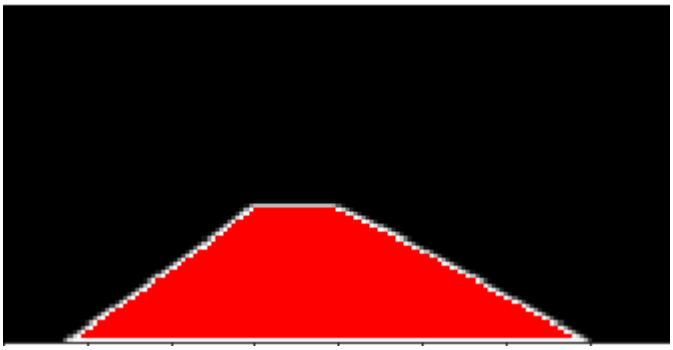
1. Rainy Morning:16%
2. Clear night:18%
3. Cloudy Afternoon:66%

Size of the data set:

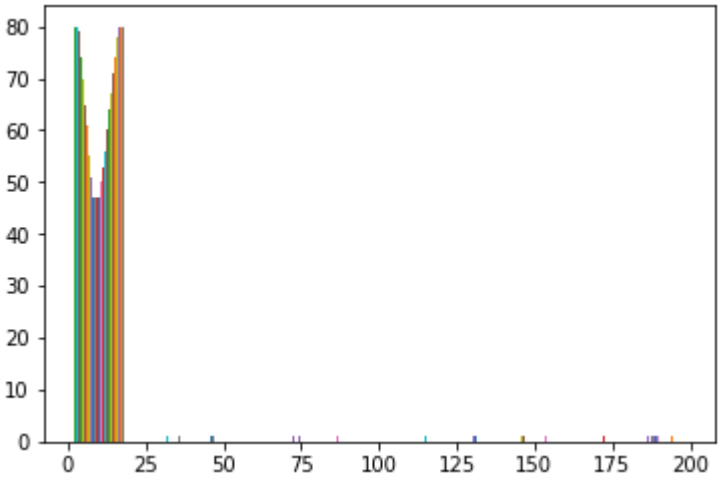
A total of 12754 images are there in the data set in which 10 images are preserved for testing. So 10% of the rest image is used for the validation and remaining is used for training.

Exploratory Visualization

The data can be visualized as the bounding box of the lane. We can consider it as the segmented image of the lane.



The histogram should be shantered near 0 and it can be visualize below:



Algorithms and Techniques

The algorithm and technique used in the project was mainly the convolutional network. CNNs use a variation of multilayer perceptron designed to require minimal preprocessing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. The convolution network is layer based architecture. The main layers of the architecture are convolution, Activation, Max pooling, up sampling.

Convolutional:

The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting. Convolution is a mathematical operation on two functions (f and g) to produce a third function that expresses how the shape of one is modified by the other.

Formula:

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

The same can be represented in descrete domain as follows:

$$\sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

Activation:

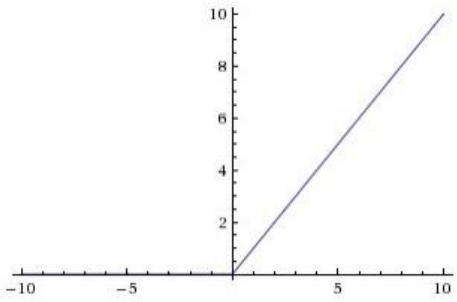
The network can be expressed by the following formula:

$$Y = \sum (weight * input) + bias$$

Here waight is multiplied with the input and bias is added and we get output. But a neuron will be fired or not is deciede by the activation function. The output can take any value so we define a range of the value in which the neuron can be fired and task is done by the activation function.

There are many activation function like sigmoid, Relu, tenh etc. Relu is used as activation function in this project. Relu activate neurons only for the positive value.The graph for Relu can be seen in the side image. Formula for the Relu is given below:

$$A(x) = \max(0,x)$$

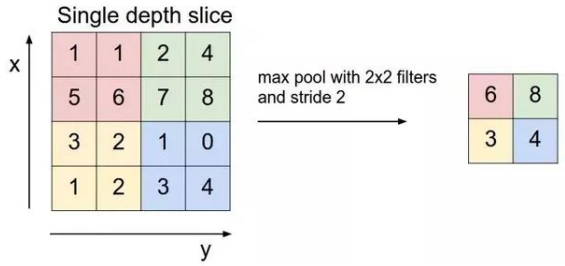


Max pooling:

Max pooling is a sample-based discretization process. The objective is to down-sample an input image, reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned.

This is done to in part to help over-fitting by providing an abstracted form of the representation. As well, it reduces the computational cost by reducing the number of parameters to learn and provides basic translation invariance to the internal representation.

Below is pictorial view of above text:



Up sampling:

Up sampling is opposite process of the max pooling it is done in order to bring the image at its original dimension. It helps the network to learn more features.

Benchmark

As discussed earlier the State of Art method is used for lane detection which does not perform well. The previous lane detection algorithm which is purely based on the image processing technique can be used for the comparison. The state of Art based lane detection method is best for benchmarking as we can clearly see the difference in the performance. The benchmarking evaluation shall be done in those conditions in which the previous model does not work well.

The output of the State of Art model is the polynomial coefficient which is used to draw the line. The color is filled in the space between the lane lines. These colors are nothing but a particular numbers.
The output of the CNN model is same. It is also the colors filled between the detected lines and these are also a particular number.
A new label for the same image will be designed which will be hand engineered by me which can be considered as the perfect output or an ideal output.
Mean squared error will be calculated for both the model (state of art and CNN) with respect the hand engineered ideal output.

The loss can be compared and can be used as evaluation matrices.

III. Methodology

Data Preprocessing

As the overview of data processing is already discussed. Here is the detailed data preprocessing methodology

Gaussian Blur: In, a Gaussian blur image processing (also known as Gaussian smoothing) is the result of blurring an image by a (named after Gaussian function mathematician and scientist). The visual effect image noise of this blurring technique is a smooth blur resembling that of viewing the through a translucent screen. Gaussian Blur is used to make the edges smoother.

Gray scaling: The images should be converted into gray scaled ones in order to detect shapes (edges) in the images. This is because the canny edge detection measures the magnitude of pixel intensity changes or gradients (more on this later).

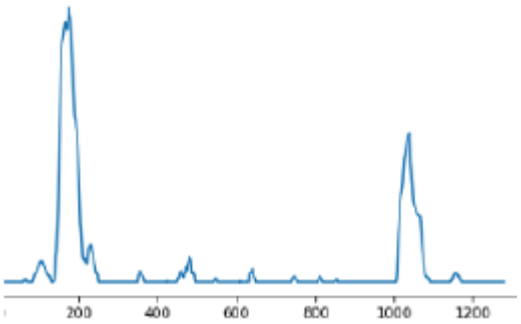
Canny edge detection: Canny edge detection which make use of the fact that edges has high gradients (how sharply image changes — for example, dark to white)

Region of interest: When finding lane lines, we are not interested in other environmental disturbances .So we mask the other regions as per our requirement.

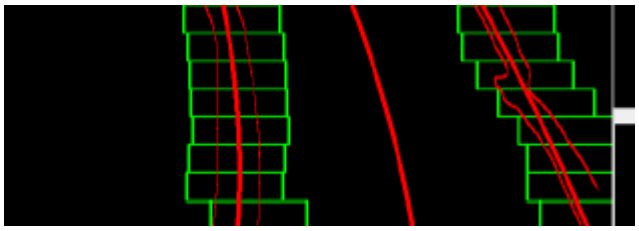
Perspective transform: We now need to define a trapezoidal region in the 2D image that will go through a perspective transform to convert into a bird's eye view. It's the change in the perspective when we see the image from side and when we see it as top-down view.
The curve lines looks almost straight after the transform as can be seen in the below image.

Histogram: We then compute a histogram of our binary thresholded images in the y direction, on the bottom half of the image, to identify the x positions where the pixel intensities are highest. Here the frame is divided into two parts, left and right sides and that is how we get the exact indices of the frame. The operation is done on both sides to detect the lane on both the sides.

As it can be seen in the below image the histogram is quite high at the position where lane is detected.



Sliding window: Since we now know the starting x position of pixels (from the bottom of the image) most likely to yield a lane line, we run a sliding windows search in an attempt to "capture" the pixel coordinates of our lane lines.
Next, we compute a second degree polynomial, via numpy's polyfit, to find the coefficients of the curves that best fit the left and right lane lines.
The same can be seen in the below image:



In this section, all of your preprocessing steps will need to be clearly documented, if any were necessary. From the previous section, any of the abnormalities or characteristics that you identified about the dataset will be addressed and corrected here. Questions to ask yourself when writing this section:

Implementation

It took a lot of ways to find the most suitable way. Many CNN was built to reach the final destination. The first CNN used batch normalization, three convolution layers, dropout , max pooling and followed by up sampling , Deconvolution , dropout and last layer was deconvolution as shown below:

batch_normalization (None, 80, 160, 3)
Conv (None, 78, 158, 8)
Conv (None, 76, 156, 16)
Conv (None, 74, 154, 32)
Dropout (None, 74, 154, 32)
Conv (None, 72, 152, 32)

Dropout (None, 72, 152, 32)
MaxPooling (None, 36, 76, 32)
Conv (None, 34, 74, 64)
Dropout (None, 34, 74, 64)
Deconv (None, 36, 76, 64)
Dropout (None, 36, 76, 64)
UpSampling (None, 72, 152, 64)
Deconv (None, 74, 154, 32)
Dropout (None, 74, 154, 32)
Deconv (None, 76, 156, 32)
Dropout (None, 76, 156, 32)
Deconv (None, 78, 158, 16)
Dropout (None, 78, 158, 16)
Deconv (None, 80, 160, 1)

The model looked fine and it was performing well but not up to the mark. The methodology for the implementation of the model was that it has been deign on the encoder and decoder basis. At first encoder model was used it consist of convolution max pooling and dropout the encoder reduces the size of the image and the filter size is increased in order to learn the features.

The decoder model contains the deconvolution up sampling and drop out the deconvolution is used to increase the size of the image as well as it learns the feature. Deconvolution is also known as transposed convolution.

Below is the detailed implementation process:

First, the image frames must be extracted from videos obtained. Next, I must process the images for use in making labels, for which I will use the same techniques I previously used in my computer vision-based model. I will find good source points (corners of the lane lines at the bottom and near the horizon line) and determine good destination points (where the image gets transformed out to) to perspective transform the image (mostly by making educated guesses at what would work best. When I have these points, I can use "cv2.getPerspectiveTransform" to get a transformation matrix and "cv2.warpPerspective" to warp the image to a bird's eye-like view of the road. Then, histograms will be computed using where the highest amount of pixels fall vertically (since the image has been perspective transformed, straight lane lines will appear essentially perfectly vertical) that split out from the middle of the image so that the program will look for a high point on the left and a separate high point on the right. Sliding windows that search for more binary activation going up the image will then be used to attempt to follow the line. Based off the detected pixels from the sliding windows, "numpy.polyfit" will be used to return polynomial functions that are most closely fit to the lane line as possible (using a polynomial allows for it to track curved lines as well as straight).

But the labels must be ensured to be prefect. To accomplish this takes the calculated labels must be drawn on the image. The polynomial was calculated for transformed image so the transformed image was used to draw the lane the polyfill function was used to fill the color between the lanes.

The image was transformed back to original image to visualize it properly. It was ensured that the labels calculated was correct.

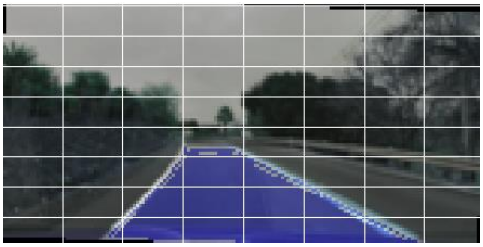
I used Keras as TesnosrFlow backend to create the convolutional neural network. At first I started with simple network and then used encoder decoder combination to accomplish the task. The last model which I found working was U –net model.

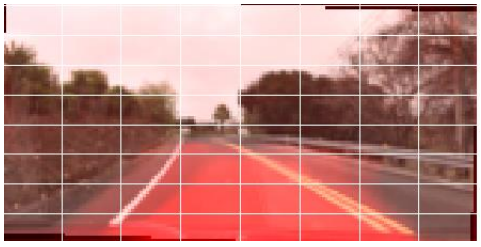
Refinement

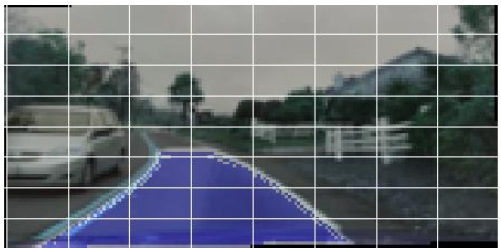
By adjusting the strides, max pooling and drop out the result became better but its performance was even worse than my previous Image processing algorithms.

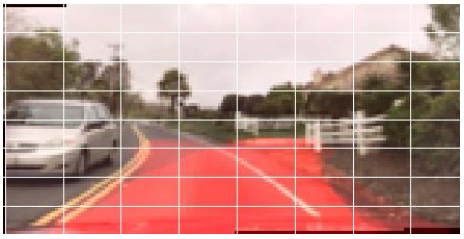
But after soome more adjustment and by lowering the trainning rate the performance increased but only for the straight road for courved road the story was still the same.

Below is the comparision of the label (in blue) and out put of CNN (in red)









I found that my problem was more related to image segmantion.I tried many networks like resnet and segnet. All were good at straight road. I thaought that I should increase more curvy road data set in the image. I reduced the straight road line so that the curvy road data set got increased. But this method also did not work well.

I thought this was time to change the network architecture again. I found Unet was used in the medical imaginary and it was very good at identifying the small curves as well as straight lines also.

The rason behind the good peformance of unet is the weight is coneected from encoder level to the decodet level which are at the same hirerchy level.

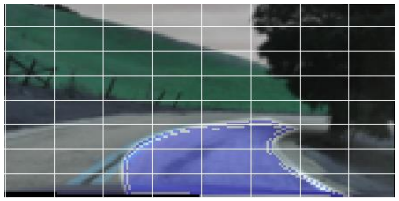
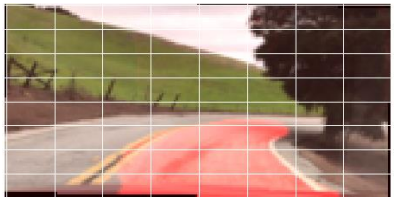
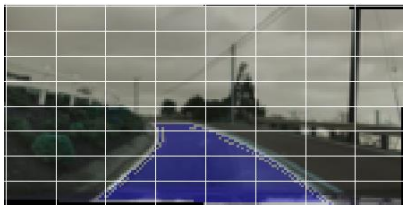
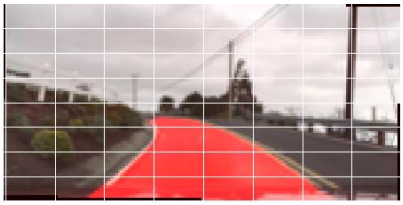
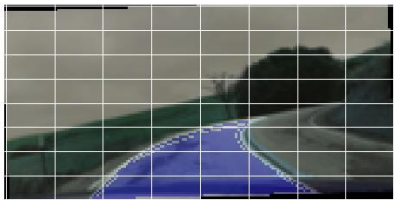
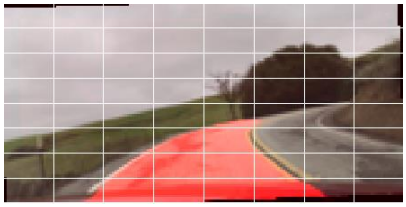
The network consists of a contracting path and an expansive path, which gives it the u-shaped architecture. The contracting path is a typical convolutional network that consists of repeated application of convolutions, each followed by a rectified linear unit (ReLU) and a max pooling operation. During the contraction, the spatial information is reduced while feature information is increased. The expansive pathway combines the feature and spatial information through a sequence of up-convolutions and concatenations with high-resolution features from the contracting path.

I used mean square error as loss function but the performance was not good even the loss decreased to 0.075.Of cousre it was detecting the lane but the red colour was overlayed to whole image.
Below is the image:

I came across the dice loss. Dice loss looks for the common port of the image. If there is any common part between the labels it coffecient is less if the common part is 0 the dice coffecient becomes 1. I had only one lebel so I had to do the dame with background.
The formula:

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|}$$

The performance was certenly increased by the Unet as it was able to detect the curve lane better than the prevoius all method below are the proof. The blue one is original leble and the red one is calculated by the network

The U net was working very much perfectly.It worked well in courved as well as in the normal road also.

The final model

So from above experiment it was found that the U net was working well. The Adam optimizer was used and for loss function dice loss was used.

The architecture of the model is as shown below:

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 80, 160, 3)	0	
conv2d_1 (Conv2D)	(None, 80, 160, 8)	224	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 80, 160, 8)	32	conv2d_1[0][0]
activation_1 (Activation)	(None, 80, 160, 8)	0	batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 80, 160, 8)	584	activation_1[0][0]
batch_normalization_2 (BatchNor	(None, 80, 160, 8)	32	conv2d_2[0][0]
activation_2 (Activation)	(None, 80, 160, 8)	0	batch_normalization_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 40, 80, 8)	0	activation_2[0][0]
conv2d_3 (Conv2D)	(None, 40, 80, 16)	1168	max_pooling2d_1[0][0]
batch_normalization_3 (BatchNor	(None, 40, 80, 16)	64	conv2d_3[0][0]
activation_3 (Activation)	(None, 40, 80, 16)	0	batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 40, 80, 16)	2320	activation_3[0][0]
batch_normalization_4 (BatchNor	(None, 40, 80, 16)	64	conv2d_4[0][0]
activation_4 (Activation)	(None, 40, 80, 16)	0	batch_normalization_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 20, 40, 16)	0	activation_4[0][0]
conv2d_5 (Conv2D)	(None, 20, 40, 32)	4640	max_pooling2d_2[0][0]
batch_normalization_5 (BatchNor	(None, 20, 40, 32)	128	conv2d_5[0][0]
activation_5 (Activation)	(None, 20, 40, 32)	0	batch_normalization_5[0][0]
conv2d_6 (Conv2D)	(None, 20, 40, 32)	9248	activation_5[0][0]
batch_normalization_6 (BatchNor	(None, 20, 40, 32)	128	conv2d_6[0][0]
activation_6 (Activation)	(None, 20, 40, 32)	0	batch_normalization_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 10, 20, 32)	0	activation_6[0][0]
conv2d_7 (Conv2D)	(None, 10, 20, 64)	18496	max_pooling2d_3[0][0]
batch_normalization_7 (BatchNor	(None, 10, 20, 64)	256	conv2d_7[0][0]
activation_7 (Activation)	(None, 10, 20, 64)	0	batch_normalization_7[0][0]

conv2d_8 (Conv2D)	(None, 10, 20, 64)	36928	activation_7[0][0]
batch_normalization_8 (BatchNor	(None, 10, 20, 64)	256	conv2d_8[0][0]
activation_8 (Activation)	(None, 10, 20, 64)	0	batch_normalization_8[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 5, 10, 64)	0	activation_8[0][0]
conv2d_9 (Conv2D)	(None, 5, 10, 128)	73856	max_pooling2d_4[0][0]
batch_normalization_9 (BatchNor	(None, 5, 10, 128)	512	conv2d_9[0][0]
activation_9 (Activation)	(None, 5, 10, 128)	0	batch_normalization_9[0][0]
conv2d_10 (Conv2D)	(None, 5, 10, 128)	147584	activation_9[0][0]
batch_normalization_10 (BatchNo	(None, 5, 10, 128)	512	conv2d_10[0][0]
activation_10 (Activation)	(None, 5, 10, 128)	0	batch_normalization_10[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 10, 20, 128)	0	activation_10[0][0]
concatenate_1 (Concatenate)	(None, 10, 20, 192)	0	activation_8[0][0] up_sampling2d_1[0][0]
conv2d_11 (Conv2D)	(None, 10, 20, 64)	110656	concatenate_1[0][0]
batch_normalization_11 (BatchNo	(None, 10, 20, 64)	256	conv2d_11[0][0]
activation_11 (Activation)	(None, 10, 20, 64)	0	batch_normalization_11[0][0]
conv2d_12 (Conv2D)	(None, 10, 20, 64)	36928	activation_11[0][0]
batch_normalization_12 (BatchNo	(None, 10, 20, 64)	256	conv2d_12[0][0]
activation_12 (Activation)	(None, 10, 20, 64)	0	batch_normalization_12[0][0]
conv2d_13 (Conv2D)	(None, 10, 20, 64)	36928	activation_12[0][0]
batch_normalization_13 (BatchNo	(None, 10, 20, 64)	256	conv2d_13[0][0]
activation_13 (Activation)	(None, 10, 20, 64)	0	batch_normalization_13[0][0]
up_sampling2d_2 (UpSampling2D)	(None, 20, 40, 64)	0	activation_13[0][0]
concatenate_2 (Concatenate)	(None, 20, 40, 96)	0	activation_6[0][0] up_sampling2d_2[0][0]
conv2d_14 (Conv2D)	(None, 20, 40, 32)	27680	concatenate_2[0][0]
batch_normalization_14 (BatchNo	(None, 20, 40, 32)	128	conv2d_14[0][0]
activation_14 (Activation)	(None, 20, 40, 32)	0	batch_normalization_14[0][0]
conv2d_15 (Conv2D)	(None, 20, 40, 32)	9248	activation_14[0][0]
batch_normalization_15 (BatchNo	(None, 20, 40, 32)	128	conv2d_15[0][0]
activation_15 (Activation)	(None, 20, 40, 32)	0	batch_normalization_15[0][0]
conv2d_16 (Conv2D)	(None, 20, 40, 32)	9248	activation_15[0][0]

batch_normalization_16 (BatchNo	(None, 20, 40, 32)	128	conv2d_16[0][0]
activation_16 (Activation)	(None, 20, 40, 32)	0	batch_normalization_16[0][0]
up_sampling2d_3 (UpSampling2D)	(None, 40, 80, 32)	0	activation_16[0][0]
concatenate_3 (Concatenate)	(None, 40, 80, 48)	0	activation_4[0][0] up_sampling2d_3[0][0]
conv2d_17 (Conv2D)	(None, 40, 80, 16)	6928	concatenate_3[0][0]
batch_normalization_17 (BatchNo	(None, 40, 80, 16)	64	conv2d_17[0][0]
activation_17 (Activation)	(None, 40, 80, 16)	0	batch_normalization_17[0][0]
conv2d_18 (Conv2D)	(None, 40, 80, 16)	2320	activation_17[0][0]
batch_normalization_18 (BatchNo	(None, 40, 80, 16)	64	conv2d_18[0][0]
activation_18 (Activation)	(None, 40, 80, 16)	0	batch_normalization_18[0][0]
conv2d_19 (Conv2D)	(None, 40, 80, 16)	2320	activation_18[0][0]
batch_normalization_19 (BatchNo	(None, 40, 80, 16)	64	conv2d_19[0][0]
activation_19 (Activation)	(None, 40, 80, 16)	0	batch_normalization_19[0][0]
up_sampling2d_4 (UpSampling2D)	(None, 80, 160, 16)	0	activation_19[0][0]
concatenate_4 (Concatenate)	(None, 80, 160, 24)	0	activation_2[0][0] up_sampling2d_4[0][0]
conv2d_20 (Conv2D)	(None, 80, 160, 8)	1736	concatenate_4[0][0]
batch_normalization_20 (BatchNo	(None, 80, 160, 8)	32	conv2d_20[0][0]
activation_20 (Activation)	(None, 80, 160, 8)	0	batch_normalization_20[0][0]
conv2d_21 (Conv2D)	(None, 80, 160, 8)	584	activation_20[0][0]
batch_normalization_21 (BatchNo	(None, 80, 160, 8)	32	conv2d_21[0][0]
activation_21 (Activation)	(None, 80, 160, 8)	0	batch_normalization_21[0][0]
conv2d_22 (Conv2D)	(None, 80, 160, 8)	584	activation_21[0][0]
batch_normalization_22 (BatchNo	(None, 80, 160, 8)	32	conv2d_22[0][0]
activation_22 (Activation)	(None, 80, 160, 8)	0	batch_normalization_22[0][0]
conv2d_23 (Conv2D)	(None, 80, 160, 1)	9	activation_22[0][0]

Total params: 543,641
Trainable params: 541,929
Non-trainable params: 1,712

IV. Results

Model Evaluation and Validation

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the model's solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:

As discussed above I started with the encoder and decoder model but the performance was not so good. It was not working well in the curved lane. The model was not performing well even in the bad weather conditions.
So I decide to use U net model and used mean square error. The problem with the mean square error was it was not suited well for segmentation.
So I decided to use dice error coefficient for the loss function. The dice error tries to reduce the common pixel between the two class. I had a lebel class background. The background itself can be treated as a calss so the model was trying to reduce the common pixcels between background and the lane.I achieved a very good performance.

Now the model was performing well in straight as well as curve road. It performed well in the bad ligting condition also.

Surprisingly it performed well even if the glass of the car was not well (second last image below)

Performance on unseen data

Below is the output of the model on unseen data.


Performance on unseen data in bad conditioned road

The model is robust enough to trust the result .

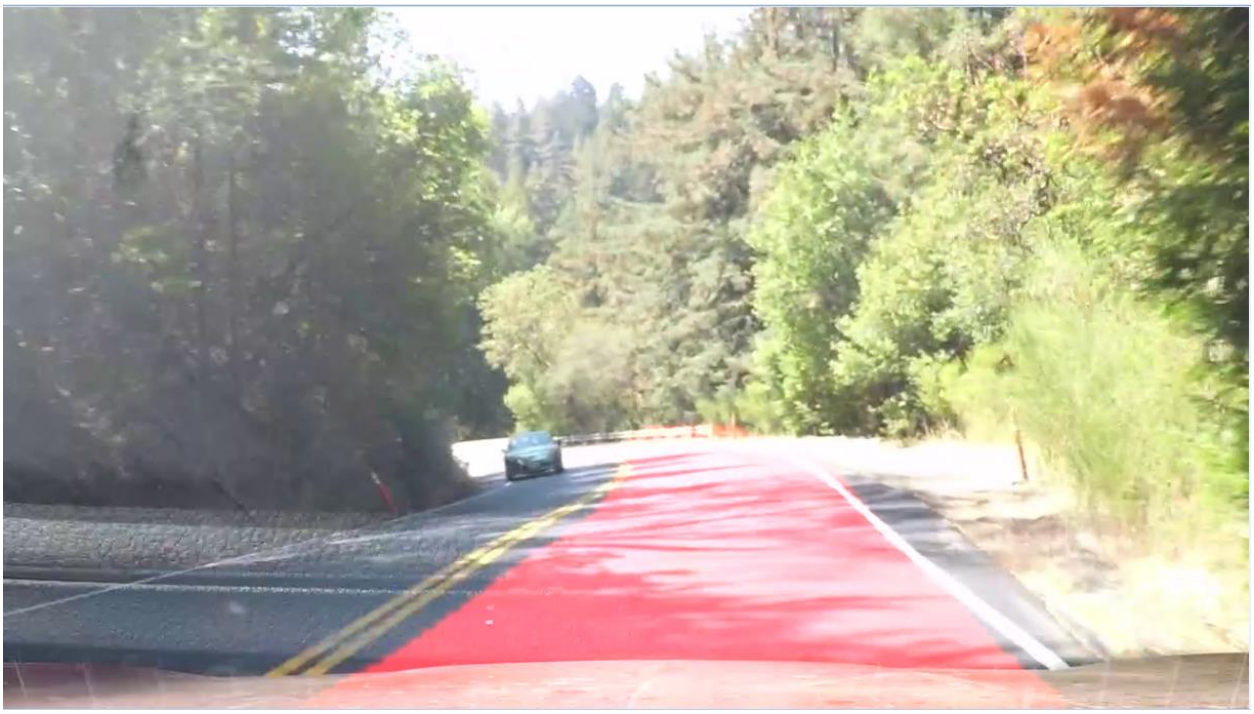
Justification

The final result obtained was much better than the benchmark. The image processing algorithm was not able to perform well in bad light conditions and in case of bad weather. The image processing algorithm was nor working fine if lane marking was not properly visible.

Form the above image we can see that the U net is working so well in all the conditions. It’s much better than the image processing algorithm.

V. Conclusion

Free-Form Visualization



We can visualize above output as the important quality of the result. Note that the image lighting condition is not good, there are a lot of shadow also on the road and if we focus on the right bottom side we fine its blur. The model is still working well.

Reflection

The project was started because of a lot of interest in the automation driving. The first challenge came to get the proper dataset. Getting a good data set can win the race but to get a good data set is a great challenge. After searching for online resources I found that the lane Images was there but it was not labeled properly.

The next challenge was to label the data. I had a great image processing algorithm but it was not sufficient to label the data as it was not working well in all the conditions. I decided to label the data using Image processing algorithm and then correct the result manually. It worked well but I was left with so much of manual work. The Matlab ground truth labeler came handy to help for some extent. I got some online tool also to label the data.

To label the data I had to process the image through various Image process algorithm. The image was gray thrashed and then the edges was detected the ROI was applied on the image and after that the bird eye view was obtained. Next histogram was calculated and the sliding window was applied on the image to find the lane. At last the polyfit function was used to get the coefficient

of the lanes. The lanes was drawn on the image and inverse perspective was calculated to obtain the final label.

Now the real challenge was ahead for which I was excited – Build a neural network model. I started with a simple model it worked but not well. I adjusted the strides and polling and other parameters I good some better result but not up to the mark.

I changed to model and started working on U net model. I MSE as loss function and Adam optimizer. Adam optimizer worked well as it not stop at the local minima and slowly it converges towards the less error. But the MSE was not doing good job. I changed the loss function to the dice loss as it tries to rescue the common pixel between the classes and it gives the result with well-defined boundary. I was happy with the result the dice coefficient was mire that 75% it performed well.

I used Keras checkpoint function. The best part of the function was that we can retrain our model and it saves only the best weights. In my case the weight was getting saved only if the dice coefficient was increased. After some more hours of training I was able to get model efficiency as 96%. Now the model was excellent at work.

The final model perfumed well as per my expectation.

Improvement

The improvement can be made by designing such model which works fast. The model architecture can be changed to achieve more efficient result.

One major improvement can be done if the model is made to detect all the lanes present on the road. The current model is detecting only ego lane.