Lane Detection with Convolutional LSTM

Sriram Shreedharan
Electrical and Computer Engineering
A59010541

Marcus Schaller
Electrical and Computer Engineering
A59004764

Abstract

Lane detection is an extremely important technique used in autonomous driving. The aim of our project is to implement lane detection for autonomous vehicles using a deep learning solution. The goal is to utilize convolutional neural network (CNN) architecture to encode and decode video frames and ConvLSTM blocks to analyze the video and predict the lane using past frame data. Once this is working on a real world dataset we hope to apply this technique on a simulator and eventually a real world small scale car.

1 Introduction

The field of autonomous vehicles has grown rapidly in the past few years and lane detection systems is an important part of it. The control of a vehicle is based on the detected lane and the vehicle's position inside the lane. It is important to detect the lane markings accurately for a running vehicle so it can stay inside the lane without causing any accidents. The aim of our project is to implement a deep neural network architecture that can take in continuous road images and detect the lane markings from these images. Since lane detection is a type of segmentation problem, we plan to combine a Convolutional Neural Network (CNN) architecture to encode and decode the image features and Recurrent Neural Network (RNN) to embed lane information from past images. The Unet Architecture has an encoder-decoder structure and is commonly used for segmentation applications, so we are using that along with the Convolutional LSTM blocks [1]. We have tested it on two datasets: TVTLane dataset and our own dataset taken by using Donkey Car Simulator. The network seems to perform well on both the datasets with high accuracy and F1 score.

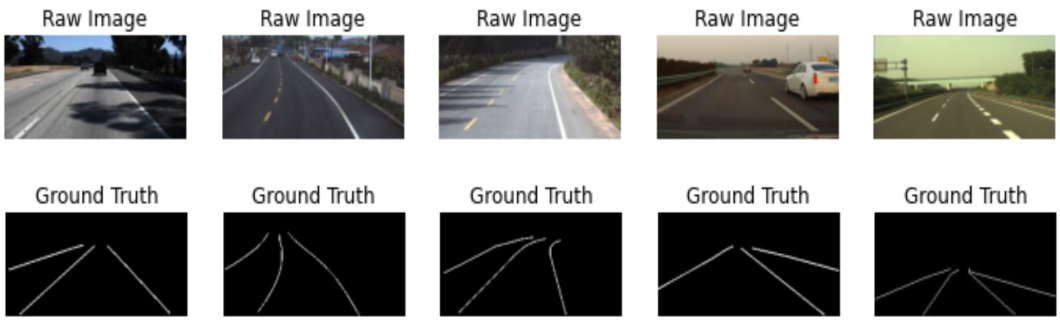


Figure 1: Sample Input Images and Groundtruth from TVTLane Dataset

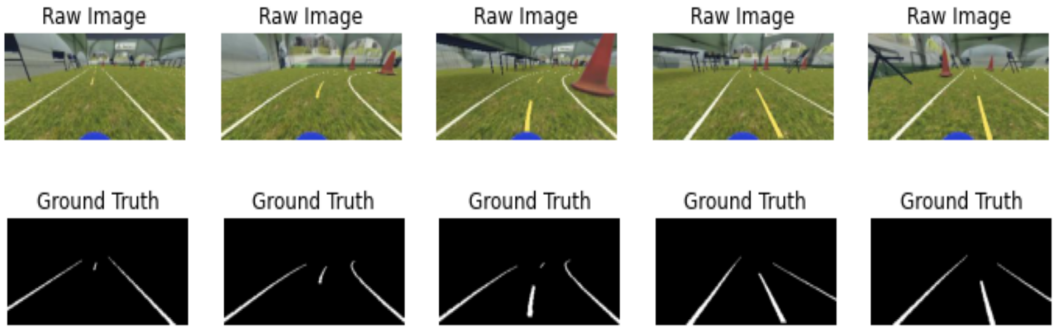


Figure 2: Sample Input Images and Ground-truth from Simulator Dataset

2 Related Work

This project we decided to work on this project as it is an important element for an imitation learning project we are working on. That project involves driving a car around a track and recording the control inputs as well as images at each time-step. Then use this data to train ResNet to imitate the control input given a camera frame input. While this has been proven to work in a simulator, it struggles to handle perturbations in a real life environment. [2] In order to handle this we plan to feed the lane prediction into the ResNet rather than original image. This requires us to evaluate a variety of methods for lane detection.

Classic methods utilize color-space thresholding and Canny detection to convert the RGB road image into a binary image. It then uses Hough lines on the binary image to estimate the lane positions. [3] However, this method is subject to error when exposed to different lightnings and occlusions. To combat this deep learning solutions have become a popular method used in detecting lanes. As this problem is a classic image segmentation problem with only a land and background class a variety of popular encoder-decoder architectures can be used. One popular network known as LaneNet uses both binary segmentation to detect the lane lines as well as instance segmentation to detect the area of each of the different lanes in an image. They then utilize an "H-net" which trains the parameters within a transformation matrix H to project the lane and predict the future lane positions using curve fitting. [4] These deep learning methods have proved to output very accurate lane segmentation's and we believe we believe once we solve this problem in our simulator domain it will be very easy to implement it in a real world R/C car.

3 Method

The architecture that we have used is shown in Figure 3.

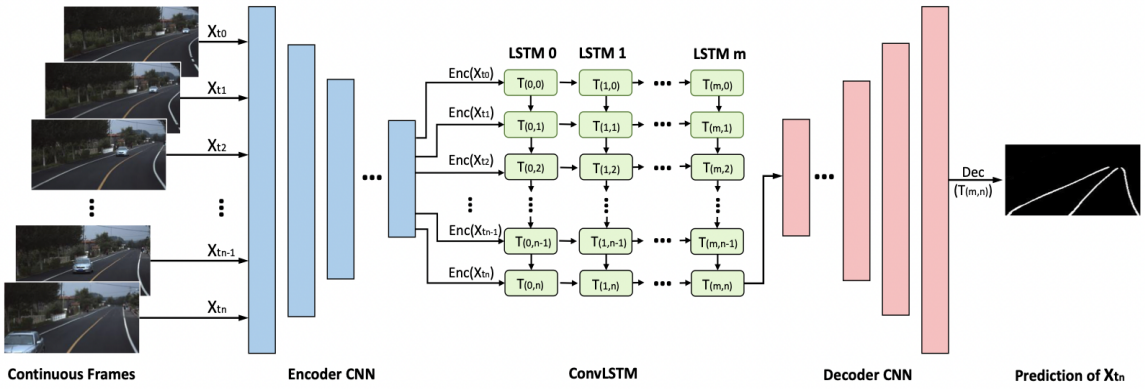


Figure 3: The UnetConvLSTM Architecture

The architecture basically combines Convolution Neural Network (CNN) and Recurrent Neural Network (RNN) for detecting the lane. Since lanes are continuous, we can get more accurate results by taking the information from past images, as it can help overcome occlusions, shadows and lighting issues in the present image. So, RNN is chosen here to process the continuous images produced by a camera in an autonomous vehicle as it works well on time-series analysis problems. For a given time, the encoder CNN takes as input the current frame and a few previous frames (we have chosen 4 frames), and outputs feature maps. This is then given to a Covolutional LSTM network to obtain lane prediction information, and finally the decoder CNN is used to output a probability map with the same size of the input image. [1]

3.1 Unet Architecture

For the encoder-decoder architecture we have used UNet [5] as shown in Figure 4 which encodes the current and previous images into feature maps.

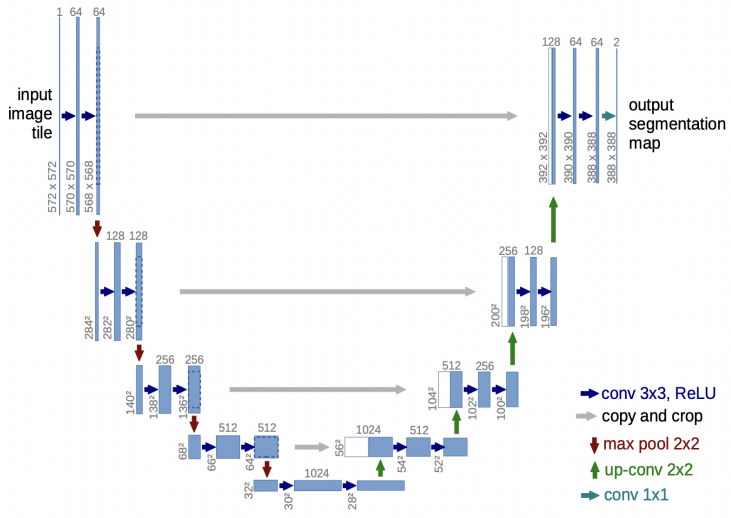


Figure 4: The Unet Architecture

The encoder architecture used in our approach (with the right dimensions) is shown in Fig. 5. It takes in a 128x256x3 input image and continuously applies convolution and maxpooling to gradually increase the feature channels and finally outputs an 8x16x512 output feature map.

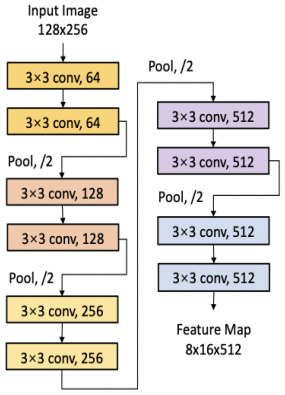


Figure 5: The Encoder network in Unet for our approach

On the decoder side, instead of downsampling we are upsampling by using up-convolution or convolution transpose. The corresponding encoder side features are also concatenated with the upsampled features so that the network has information about both input and output features at every stage. Finally, the decoder outputs an image with the same size as input image and the number of channels equal to the number of possible classes.

3.2 Convolutional LSTM

LSTM works well with time-series data and has been used widely to embed information from past data, however it doesn't work that well for spatial data due to redundancy. By substituting the fully connected layers in the LSTM block with Convolutional layers [6], the network can work better for spatiotemporal problems. This is the reason for choosing Convolutional LSTM layer to process the images from previous frames. The Convolutional LSTM cell is shown in Fig. 6.

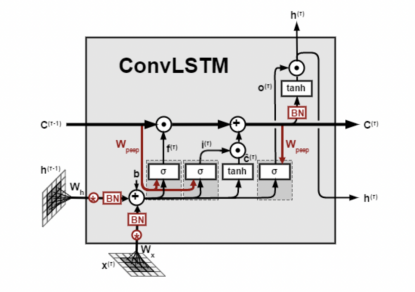


Figure 6: A ConvLSTM Cell

The equations of ConvLSTM are shown below:

$$i_t = \sigma(W_{xi} * X_t + W_{hi} * h_{t-1} + W_{ci} \circ c_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf} * X_t + W_{hf} * h_{t-1} + W_{cf} \circ c_{t-1} + b_f)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ tanh(W_{xc} * X_t + W_{hc} * h_{t-1} + b_c)$$

$$o_t = \sigma(W_{xo} * X_t + W_{ho} * h_{t-1} + W_{co} \circ c_t + b_o)$$

$$h_t = o_t \circ tanh(c_t)$$

where * denotes the convolution operation, $\circ$ denotes the Hadamard operator, c_t is the cell output at time t, h_t is the hidden state at time t, and i_t, f_t, o_t are the states of input, forget and output gate at time t.

The output from the encoder for all 5 frames are given to 2 layers of Convolutional LSTM cells which takes in an input of size 5x8x16x512 and outputs the same. Then the last output image (which is the current frame) is alone taken and is sent through the decoder network which finally outputs an image with the same size as input image and probabilities for the two classes (128x256x2).

3.3 Training and Testing

The network is first trained on the TVTLane dataset with cross entropy loss and a learning rate of 0.01. The learning rate is updated using Reduce LR on Plateau scheduler on Pytorch which reduces the learning rate when the training loss doesn't decrease any further. The input to the network is the current image for which the lane needs to be detected and 4 past images. The training is done with a batch size of 1 on the UCSD datahub (higher batch size gave CUDA memory error). The weights after training with the TVTLane dataset are stored and is then used for training with the simulator dataset. We just needed to train for 10 epochs with the simulator dataset and we obtain a high test accuracy and F1 score.

4 Experiments

We are using two datasets for our project (1) TVTLane Dataset from Wuhan University and (2) our own Simulator Dataset from Donkey Car Simulator. As there are many images they are included at the end of the report.

4.1 TVTLane Dataset

The dataset contains 19383 image sequences (each sequence has 20 frames) with 39460 labeled frames for lane detection. Each image is of size 128x256 and the groundtruth label is a binary image with white pixels to denote the lane marking and black pixel otherwise. The 13th and the 20th frame within each sequence is labeled. The training set is augmented with flipping and rotation as well. In this dataset, the broken lane markings in the middle of the road are also considered continuous and the groundtruth is labeled accordingly. [7]

While training, different sets of previous frames are provided as input. For the 13th frame, we train with the frames [1,4,7,10,13], [5,7,9,11,13], and [9,10,11,12,13] and for the 20th frame, we train with the frames [8,11,14,17,20], [13,15,17,19,20], and [16,17,18,19,20]. While sampling the previous frames differently does not affect the performance of the network while testing, it gives us more data to train on.

4.2 Simulator Dataset

We also created our own dataset using the donkey car simulator. The simulator environment consists of a RC car and a track with lanes to drive. The car is also equipped with a camera and images of the environment can be taken. We drove the car for several minutes in the simulator track and collected 100 image sequences i.e 2000 images in total. We split 60 sequences for training, 20 each for validation and testing. We labeled the 13th and 20th frame for each sequence like in TVTLane. Using traditional OpenCV methods we were able to easily create the ground truth images since the environment is evenly lit. The flow for the OpenCV methods goes as follows. First the images are converted to hue, saturation, lightness color space. Then sobel edge detection is used on the lightness channel to detect sharp discontinuities in the pixel intensities along the x and y axis. Then the image is Gaussian blurred and thresholded on saturation values and original red values. Finally the HSL filtered and RGB filtered color spaces are combined with some additional filtering to remove any extra pixels.[8] Unlike the TVTLane dataset, we did not consider the broken lane lines as continuous in our ground-truth label.

The training is done similarly to the TVTLane by sampling from different sets of previous frames. The training loss, validation loss and validation accuracy graph is shown in Fig 7. We trained on two different networks: one is just the Unet architecture which takes in a single image as input and outputs lane detections, the other is the UnetConvLSTM architecture which takes in the current frame along with the previous frames and outputs the lane detections. The two networks have almost similar performance when tested on the simulator dataset which is expected as the simulator dataset has no occlusions or lighting variations. However, on the TVTLane dataset output as shown in Fig. 8, we can see that the Unet architecture doesn't work that well in poor lighting conditions and occlusions. On the other hand, we can see that the UnetConvLSTM works really well even in those situations. So, even though there isn't much effect due to the addition of the Convolution LSTM block right now on our simulated RC car, we believe it will make a difference when we transfer it to a real world RC car (which we are planning to do next!).

4.2.1 Performance Metrics

To quantitatively evaluate the performance of the network, we calculated 4 metrics that tells us how good the segmentation works. Those are validation accuracy, precision, recall, and F1 score. The accuracy is not a great metric since this is a highly disproportionate binary classification problem (number of black pixels is far greater than number of white pixels). So even if we classify the image as all black we will still get an accuracy greater than 95% and that's why it's important to evaluate the performance using other metrics such as the precision, recall and F1 score. The formulas for these 4 metrics are as given below:

$$Validation\ \ Accuracy = \frac{True\ \ Positive + True\ \ Negative}{TotalNumberofPixels}$$

$$Precision = \frac{True\ \ Positive}{True\ \ Positive + False\ \ Positive}$$

$$Recall = \frac{True\ \ Positive}{True\ \ Positive + False\ \ Negative}$$

$$F1\ \ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

where $True\ \ Positive$ denotes the lane pixels classified correctly, $False\ \ Positive$ denotes the non-lane pixels classified incorrectly, $True\ \ Negative$ denotes the non-lane pixels classified correctly, $False\ \ Negative$ denotes the lane pixels classified incorrectly.

The Table 1 shows these metrics for both Unet and UnetConvLSTM architecture on the TVLane dataset and simulator dataset.

Table 1: TVT Dataset Results

Technique	Test Accuracy	Precision	Recall	F1-Measure	Running Time
Classic CV	95.4	0.084	0.267	0.127	0.0023
UNet	96.54	0.790	0.985	0.877	0.012
UNet_ConvLSTM	98.47	0.857	0.958	0.904	0.0285

Table 2: Simulator Dataset Results

Technique	Test Accuracy	Precision	Recall	F1-Measure	Running Time
UNet	99.88	0.96	1.0	0.98	0.0132
UNet_ConvLSTM	99.90	0.97	1.0	0.98	0.0291

5 Limitations and Future Work

As of right now we have not tested this network on a Jetson Nano as it has to be converted from PyTorch to tensorRT since PyTorch is too memory intensive. If we find that it runs fine in real time no additional work will be needed for the segmentation task. However if it still struggles we will look into using GRU or Transformers architectures since they often run better on lightweight platforms. Next it is likely we will have to make a custom dataset for the real life images to additionally train our model with. Once we have the lanes segmented we will then be able to apply imitation learning with the segmented images.

6 Supplementary Material

Video Presentation (8 min long): https://youtu.be/ZcJJwUDCfe8

Sped up Presentation (under 5 min): https://youtu.be/Yd7TS0jsiB8

References

[1] Qin Zou et al. “Robust Lane Detection From Continuous Driving Scenes Using Deep Neural Networks”. In: *IEEE Transactions on Vehicular Technology* 69.1 (Jan. 2020), pp. 41–54. DOI: 10.1109/tvt.2019.2949603. URL: https://doi.org/10.1109%2Ftvt.2019.2949603.

[2] *TriTorch TriTorch*. https://github.com/Triton-AI/TriTorch. Accessed: 2022-05-01.

[3] Abdulhakam.AM. Assidiq et al. “Real time lane detection for autonomous vehicles”. In: *2008 International Conference on Computer and Communication Engineering*. 2008, pp. 82–88. DOI: 10.1109/ICCCE.2008.4580573.

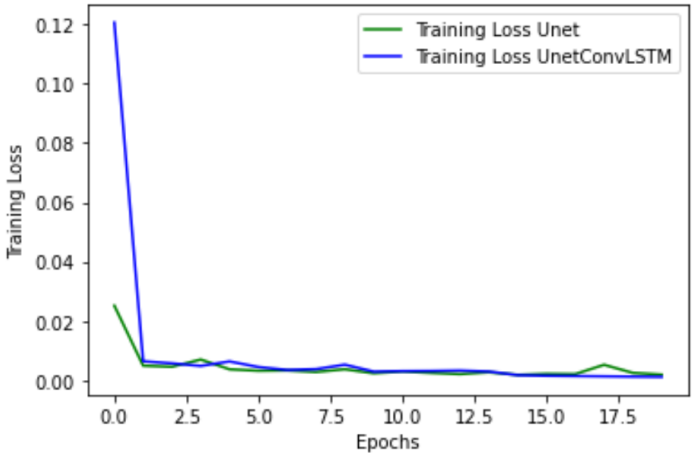
[4] Davy Neven et al. “Towards End-to-End Lane Detection: an Instance Segmentation Approach”. In: June 2018, pp. 286–291. DOI: 10.1109/IVS.2018.8500547.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. DOI: 10.48550/ARXIV.1505.04597. URL: https://arxiv.org/abs/1505.04597.

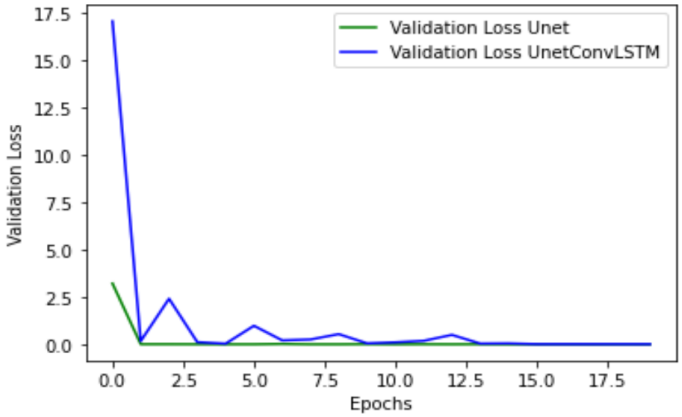
[6] Xingjian Shi et al. *Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting*. 2015. DOI: 10.48550/ARXIV.1506.04214. URL: https://arxiv.org/abs/1506.04214.

[7] *tvtLANE tvtLANE*. http://mvr.whu.edu.cn/codedata/index.html. Accessed: 2022-05-01.

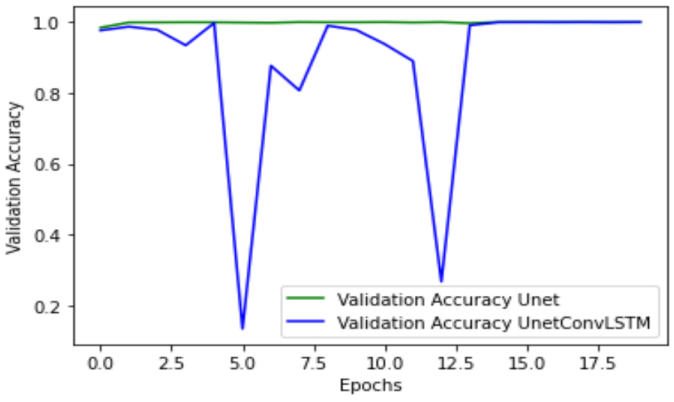
[8] $Lane_Detection Lane_Detection$. https://github.com/minyoung-chang/Lane_Detection. Accessed: 2022-05-01.



(a) Training Loss



(b) Validation Loss



(c) Validation Accuracy

Figure 7: Loss and Accuracy Plot for the Simulator Dataset

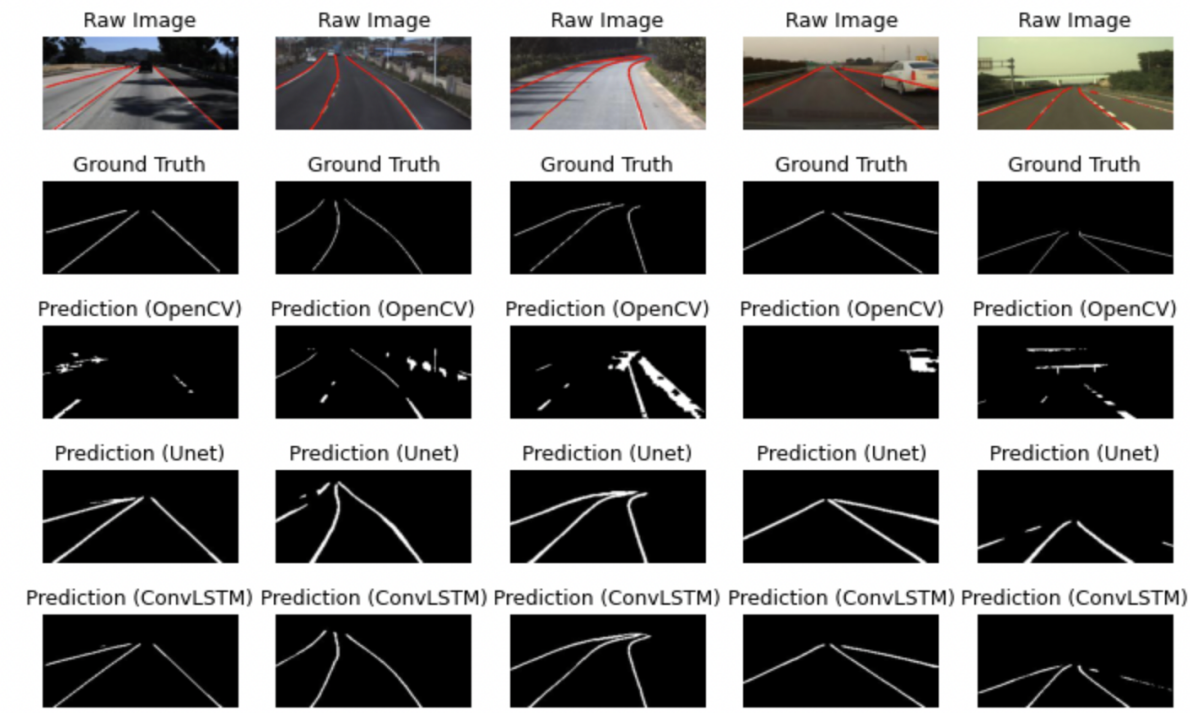


Figure 8: The Output Prediction of OpenCV, Unet and UnetConvLSTM on a sample TVTLane Dataset

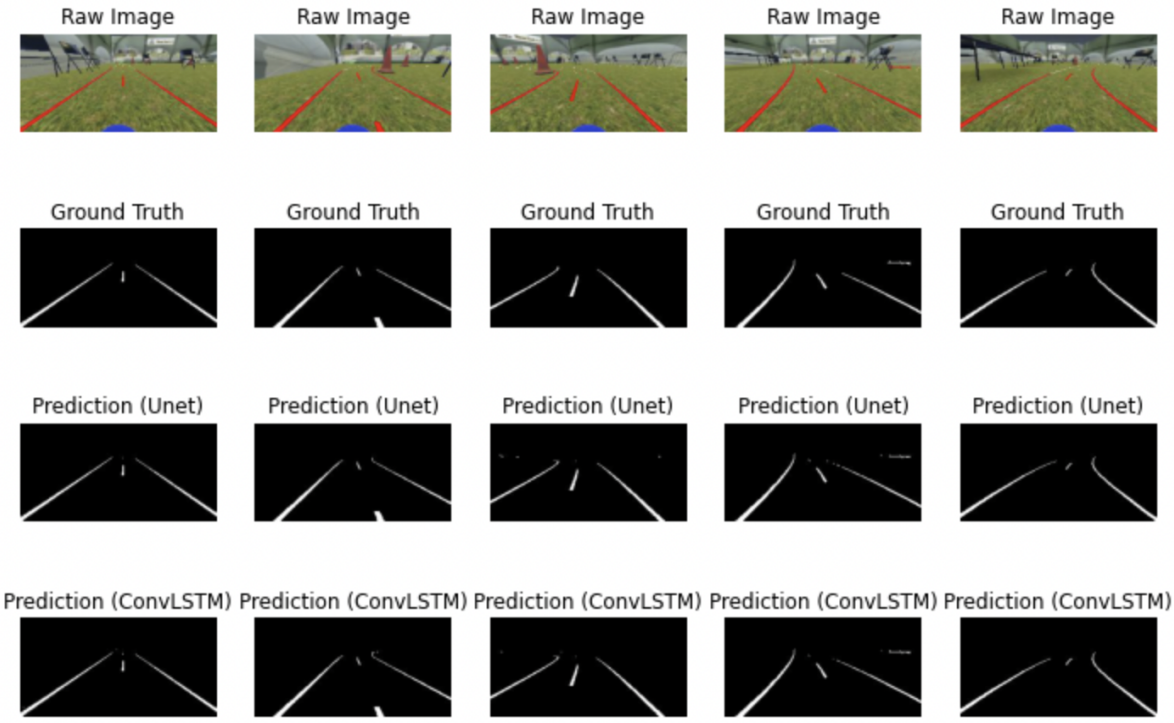


Figure 9: The Output Prediction of Unet and UnetConvLSTM on a sample simulator Dataset