DEEP LEARNING-BASED LANE DETECTION IN FOGGY WEATHER CONDITIONS

A PREPRINT

Pranava Swaroopa
Dept. of Automotive Engineering
CU-ICAR
pswaroo@clemson.edu.

Sanskruti Jadhav
Dept. of Automotive Engineering
CU-ICAR
sanskrj@clemson.edu.

Siddharth Joshi
Dept of Mechanical Engineering
Clemson University
sajoshi@clemson.edu.

May 7, 2022

ABSTRACT

Vehicles are increasingly offering advanced driver assistance in the form of lane keep assist, emergency brake assist, and many other things to drive safer. Many vehicles like Teslas depend on camera image feed to understand their environment. Since a camera is cheaper and easier to adapt compared to other advanced sensors like LiDAR or RADAR. Lane-keep assist systems based on cameras need a robust drivable area detection algorithm that can consistently perform well in inclement weather conditions like cloudy, rainy, snowy, and foggy. We use a simple encoder-decoder type convolution neural network to perform lane detection. We present our results from training on the state-of-the-art BDD100K dataset. We achieved a mIOU of 44.15% while only using worse weather condition images

Keywords CNN · BDD100K · Drivable Area

1 Introduction

Like human drivers, self-driving vehicles have trouble seeing under inclement weather. Many autonomous cars rely on lidar technology to supplement, which works by bouncing laser beams off surrounding objects to give a high-resolution 3D picture on a clear day but does not do so well in foggy and other weather conditions. Camera performance suffers from uneven lighting conditions or snow-covered signs and lane markings. Driving in foggy weather conditions has been a significant safety concern for many years due to poor driver lane-keeping performance [10].

Autonomous vehicular navigation that uses visible light has one chief obstacle, an inability to handle misty driving conditions. Most cars rely on Advanced Driver Assistance Systems (ADAS), which offers lane keep assist and steering assist. The core use cases of this lane detection technology are the ADAS driving applications like Lane-keeping assist, Lane Departure Warning, Lane Change Assistance, and Adaptive Cruise Control. Deep learning algorithms can help in these conditions by generating accurate predictions. The primary aim of these algorithms is to provide accurate predictions of lanes even in adverse weather.

With the low visibility of roads due to the cloudy, rainy, snowy, foggy climate, camera cannot capture a clear image of roads. In worse weather, light that reaches the camera's sensor will have been reflected by airborne water droplets and be affected by other natural elements, which may not be representative of the actual environment. We attempt to use a dataset that captures these realistic scenarios and train a deep learning model using convolutional neural network frameworks to predict Drivable Area in inclement weather conditions.

2 Literature Review

As the automotive industry progresses towards full-scale autonomous driving, developing accurate methodologies that function in varied weather conditions [7]. While classical approaches of image processing have been applied for this

initially, the rise of convolution neural nets has accelerated the development and deployment process in the industry. Many cars now come with lane departure warning systems that alert the driver when the car is unintentionally crossing the lane markings. This system totally depends on the front camera fixed on the front of the car; This has driven the research to find a robust lane prediction algorithm that can accommodate all types of weather conditions.

Autonomous car companies such as Tesla, Waymo, Mercedes Benz have faced their fair share of speed bumps while tackling the issues with navigating in harsh weather. A Professor at UC San Diego [1] has used an inexpensive approach of fusing radar and lidar as radars are cheap. His team conducted tests using simulation of clear days and nights and scenarios where there is a lot of fog. The lidar with radar system performed better than the lidar alone system. NVIDIA lab is using AI with the aim to improve autonomous vehicle perception. They have trained a Deep Neural Network to detect moving and stationary objects as well as accurately remove the false detections.

Tesla is the only company that uses a Perception system made only with cameras called “Tesla Vision". Tesla Model S has 8 cameras which provide a 360^{o} view of the environment. Other than cameras, ultrasonic sensors are used to provide parking assistance. Tesla uses its own developed neural network called Hydra Net.

All 8 images are first processed by image extractors. To do so, architectures similar to ResNets are used. Then, there is a multi-cam fusion. The idea is to combine all 8 images into a super-image. For that, the HydraNet uses a Transformer-like architecture. Then, there is a time-fusion.The idea is to bring time into the equation and fuse the super-image with all previous super-images. For that, there is a video queue of N images. For example, if they want to fuse within 2 seconds, and given that the cameras work at 36 Frame Per Second, N would be 72.The time fusion is done with 3D CNNs, RNNs, and/or Transformers. Finally, the output is split into HEADS.Each head is responsible for a specific use case and can be fine-tuned on its own. If the heads used to be simply heads, they now have trunks and terminals. It is a way to get deeper and more specific on the type of use case. However, Tesla revokes control of the vehicle during adverse conditions and asks the driver to take control of the steering wheel. There are signals produced by radar and camera sensors that notify the driver of poor weather conditions.

The ’Waymo Driver,’ the autonomous driving suite from Waymo, approaches the problem of inclement weather conditions by using a combination of things. In their perception module, the sensing of the world is done by combining the inputs from the camera, lidar, and radar. Sensor fusion is powerful in solving complex environments, lanes can only be detected using cameras. And this would mean the car is starting to rely on other things to follow lanes. They also upgraded their sensor systems by using microwaves [2] instead of light in their radar, which can travel through the fog and not have hindered visibility under harsh weather conditions. During dense foggy conditions, the absorption, scattering and reflectivity off the surfaces is reduced, thereby affecting the performance of LiDAR effectively. If rain occurs in such conditions the effective range of LiDAR is further drastically reduced. Thereby making radar and camera important during adverse weather conditions.

Mercedes Benz uses sensor fusion of radar and camera sensor for their active lane assist feature. This ADAS feature mainly uses the camera to detect the boundary lines of the lane in conjunction with radar to check areas to the side and rear of the car. The December 2020 feature consists of primary weather and road conditions that can hinder the functionality of lane-keeping assist included in the Mercedes Benz 2014-2020 CLA-Class, 2011-2019 B-Class MPV, and 2014-2020 B-Class. The issue talks about several environmental factors that can cause poor sensor performance in times of heavy fog or rain. These implicit errors are thus a weak point that needs continuous improvement through Deep learning methods [11]

Lane detection suffers as the road illuminations change, when there is damage on the lane markings and when the weather affects visibility. A review published in the IEEE journal [9]identified the research gap in existing techniques. In [3], they propose HSV color space-based detection algorithm that uses Progressive Probabilistic Hough Transform to detect lane and lane markings. This method does not perform well under foggy conditions and when the lane markings are not in good condition. In [7], the images are classified and processed differently according to their illuminations. According to the whole gray mean edge detection filters are applied to extract lane pixels.

On the other hand, many neural networks have been applied to [9] predict lane markings. Using neural networks increases the cost of processing time in real-time applications. We see that many networks have chosen datasets that are made mostly of cityscape data that involve minimum instances of harsh weather conditions. The [13] describes a robust Deep Convolutional Neural Net that is a Spatial-Temporal Method with segmentation and structural analysis. This is the latest effort in the lane detection algorithms and has robust performance against weather conditions but suffered under disturbances to the camera and when the lane markings were too worn out.

Lane-DeepLab is one of the methods that addresses this problem, it is a lane segmentation detection method that applies the DeepLab3+ state of the art deep learning model for semantic image segmentation model. The fully convolutional network FCN assigns semantic labels at the pixel level using an end-to-end encoder decoder network architecture. The backbone of this architecture is ResNet101 for feature extraction and atrous spatial pyramid pooling ASPP is employed

in the encoder part. This technique down-samples the spatial resolution of the input, developing lower-resolution feature maps and up-samples the feature representation to fully resolution segmentation map. The down sampling is developing lower-resolution feature mappings which are learned to be highly efficient at discriminating between classes of the image data [5]. This method involves up-sampling the resolution of the feature map whereas pooling operations involve down-sampling of the resolution by summarizing a local area with a single value. This can be averaging or argmax value in the given area of pixels. Another way of down sampling is transpose convolution which involves using a single value from a low-resolution feature map and multiplying by the weights in our filter by this value, projecting these weighted values on the output feature map. The encoder network commonly used is 13 convolutional layers in the VGG16 network. Lane-DeepLab adds an attention mechanism to the encoder part, redesign the ASPP module, and name it attention atrous spatial pyramid pooling (ARM-ASPP). After which the method uses the feature fusion method in the decoder module to combine the high-level and low-level semantic information to obtain more abundant features. Additionally employing the Semantic Embedding Branch (SEB) to combine high-level and low-level semantic information to obtain more abundant features using Single Stage Headless module abundant features could be extracted, thus making the lane detection technique more robust.

SegNet is an improved model of deep convolutional encoder-decoder architecture for image segmentation. This core trainable segmentation engine consists of the encoder-decoder network as mentioned above with an additional pixel-wise classification layer. The novelty of SegNet lies in the way the decoder up-samples its lower resolution input feature map. The decoder uses pooling indices computed in the max-pooling step of the corresponding encoder to perform non-linear upsampling, eliminating the need for learning to up sample. The upsampled maps are sparse and are then convolved with trainable filters to produce dense feature maps. In comparison with the widely adopted FCN and with the well-known DeepLab LargeFOV, DeconvNet architectures. SegNet provides a good memory versus accuracy trade-off involved in achieving good segmentation performance. SegNet is motivated by scene understanding applications. Hence, it is designed to be efficient both in terms of memory and computational time during inference. It is also significantly smaller in the number of trainable parameters than other competing architectures. There is a controlled benchmark of SegNet and other architectures on both road scenes and SUN RGB-D indoor scene segmentation tasks [4].

3 Methodology

3.1 Computational Platform and Package Dependencies:

To work with such a huge amount of data it is not possible to do it locally on your machine. We used the Palmetto Cluster from Clemson University for the compute requirements. Our CONDA environment consisted of Python 3.6 with Tensorflow 2.5 with compatible OpenCV. We used the CUDA- 11.0.3, and cuDNN - 8.0 modules for using the Tesla V 100 GPU. We used a minimum of 28 Cores and 125 GB of RAM.

3.2 Dataset

The dataset used for model inference and analysis is BDD100K dataset 2020 [12], it consists of 100k images divided into training, validation and test dataset of 70000/20000/10000 ratios respectively. The images in the dataset are of high resolution (720 x 1280). Since the scope of our project was purely based on inclement weather, with the help of the label attributes we created a curated dataset containing images from inclement weather conditions namely rainy, partly cloudy, foggy and snowy. The labels of these images are in three formats namely lane lines, masks, polygon and colormaps. We found the colormaps format of drivable area segmentation more relevant to our case. We extracted the foggy, rainy, snowy and partly cloudy images and labels from the entire dataset and the corresponding drivable area labels from the main BDD100K data. The final curated dataset was a total of 15630 images of 720X1280 pixels as shown in Fig. 1.

3.3 Data Preprocessing

The curated dataset was huge in size and number which was a bottleneck with respect to time taken to process and then the data into the model. Hence due to computational constraints we decided to preprocess data separately using more than or equal to 24 cores in palmetto cluster and then using GPU based nodes for training the model. When using the Image at its full resolution and passing through the neural network, the computation of numeric values becomes more complex. Hence, we opted to preprocess the dataset, we used two different factors for downsizing the images, one that is 40% of original and 60% size of images.

After having downsized into two types of image sizes, we converted all the images into arrays using numpy and saved these arrays for further loading and preprocessing. This step involves a lot of dead kernel issues, due to the heavy size of 15630 large sized images. Additionally on using Pickle5 library to save the image arrays, we observed that

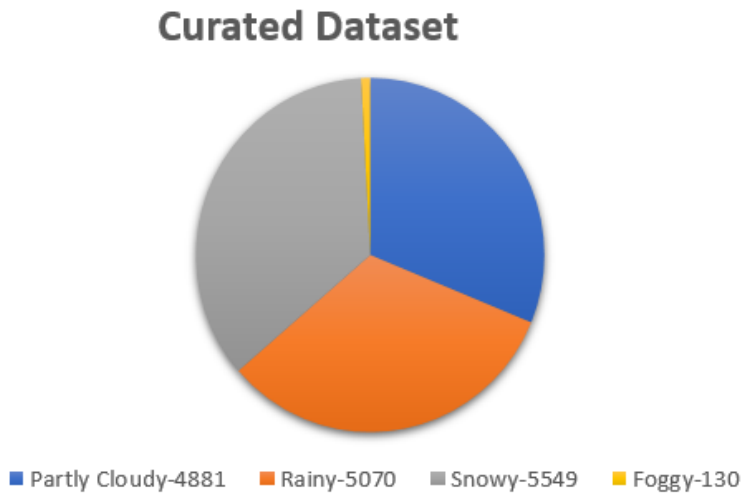

Figure 1: Distribution of the dataset

the images are not stored in the same sequence feeded to the algorithm. This would be an issue especially when the sequential nature of images and its labels needed to be preserved. After visualizing the images and labels in the saved pickled files we observed that the unique sequential nature of the dataset is lost and hence we switched back to using for loops and saving using numpy arrays.

Additional preprocessing of the labels was carried out due to the dual nature of BDD100k colormaps labels Fig. 2.

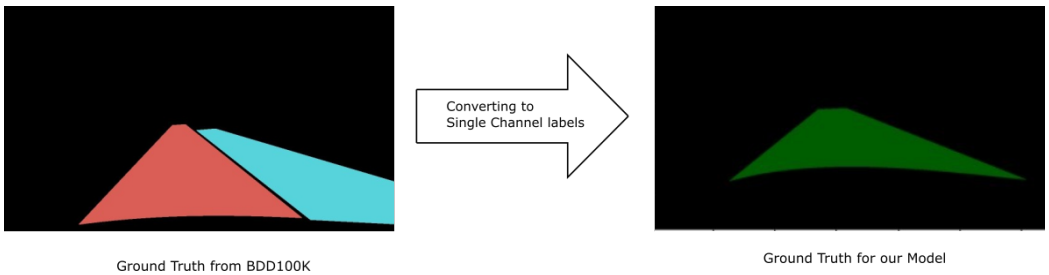

Figure 2: Ground Truth Labels after Data Preprocessing

The normalizing of the sets of resized image arrays by dividing 255 across the arrays was another complex task due to huge image matrix sizes. Hence the normalization of the datasets was done using mini batches of the data and then concatenating the individual mini batches as one huge array of 15630 x Image_Height x Image_Width X RGB channels. The overall image sizes are in Table 1:

Dataset original size	**Image sizes**	**Array Size**
15630,720,1280,3	60% reduced - 432,768,3	Images -15630,432,768,3 Labels-15630,432,768
15630,720,1280,3	40% reduced - 288,512,3	Images -15630,288,512,3 Labels-15630,288,512

Table 1: Dataset Size Details

In the final preprocessing of this curated dataset, we split the data into a training and testing set with 80-20 ratio. This ratio of image sizes for testing and training was fixed for all experiments.

3.4 Network Architecture

Our choice of deep learning architecture is a simple CNN. Since our interest lies in detecting drivable areas we researched for segmentation techniques in deep learning. Architecture discussed in [6] and other works indicate that a CNN in Encoder-Decoder framework is the choice for segmentation tasks.

We initially used a reference network design [8] to start the data preprocessing and understanding the complexities. Initial neural network used a 80 x 160 pixel image size as input and contained 15-Layers where the first layer was the BatchNormalization layer. Network is symmetric in shape and the first seven layers are 2D Convolutional layers and are Max-pooled three times. This reduces the size of the image which is later upsampled and convolved using 2D Transpose layer to output a single channel image of the same size as the original image. Based on the performance on our dataset we moved to further experiments.

3.4.1 Initial Model

Layers Description: Based on the network we referred to, we initially developed a network of 15 layers. Which consists of 5 Convolution Layers, 6 Deconvolution layers, 2 Max pooling layers, and 2 Up-sampling layers. After each convolution and deconvolution layer a DropOut of 0.2 has been added Parameters listed in Table 2.

Hyperparameter	**Values**
Image Size	40% of Original
Training Data	12504
Testing Data	3126
Batch Size	8
Epoch	10
Optimizer	ADAM (LR = 0.001)
Loss Function	Mean Squared Error
Activation Function	ReLU

Table 2: Initial Model Parameters

Model Training: After 10 epochs, which took around 10 minutes to run, the model finished with Accuracy of 0.88 , Validation loss of 0.11 and Mean IoU of 0.41 as seen in Fig 3.

```
Epoch 1/10
1172/1172 [==============================] - 80s 68ms/step - loss: 0.1169 - accuracy: 0.8831 - val_loss: 0.1165 - val_accuracy: 0.8835
Epoch 2/10
1172/1172 [==============================] - 63s 54ms/step - loss: 0.1162 - accuracy: 0.8838 - val_loss: 0.1165 - val_accuracy: 0.8835
Epoch 3/10
1172/1172 [==============================] - 62s 53ms/step - loss: 0.1176 - accuracy: 0.8824 - val_loss: 0.1165 - val_accuracy: 0.8835
Epoch 4/10
1172/1172 [==============================] - 62s 53ms/step - loss: 0.1179 - accuracy: 0.8821 - val_loss: 0.1165 - val_accuracy: 0.8835
Epoch 5/10
1172/1172 [==============================] - 62s 53ms/step - loss: 0.1158 - accuracy: 0.8842 - val_loss: 0.1165 - val_accuracy: 0.8835
Epoch 6/10
1172/1172 [==============================] - 62s 53ms/step - loss: 0.1166 - accuracy: 0.8834 - val_loss: 0.1165 - val_accuracy: 0.8835
Epoch 7/10
1172/1172 [==============================] - 62s 53ms/step - loss: 0.1168 - accuracy: 0.8832 - val_loss: 0.1165 - val_accuracy: 0.8835
Epoch 8/10
1172/1172 [==============================] - 61s 52ms/step - loss: 0.1171 - accuracy: 0.8829 - val_loss: 0.1165 - val_accuracy: 0.8835
Epoch 9/10
1172/1172 [==============================] - 61s 52ms/step - loss: 0.1176 - accuracy: 0.8824 - val_loss: 0.1165 - val_accuracy: 0.8835
Epoch 10/10
1172/1172 [==============================] - 61s 52ms/step - loss: 0.1155 - accuracy: 0.8845 - val_loss: 0.1165 - val_accuracy: 0.8835
```

Figure 3: Runtime results from training

Model Results: After training the model we used the model.predict() function to compare the actual label (Ground Truth) against the label generated from our model. As it is evident from Fig 4., the initial model does not perform well as compared to the ground truth. Hence we performed hyperparameter tuning to best fit the model.

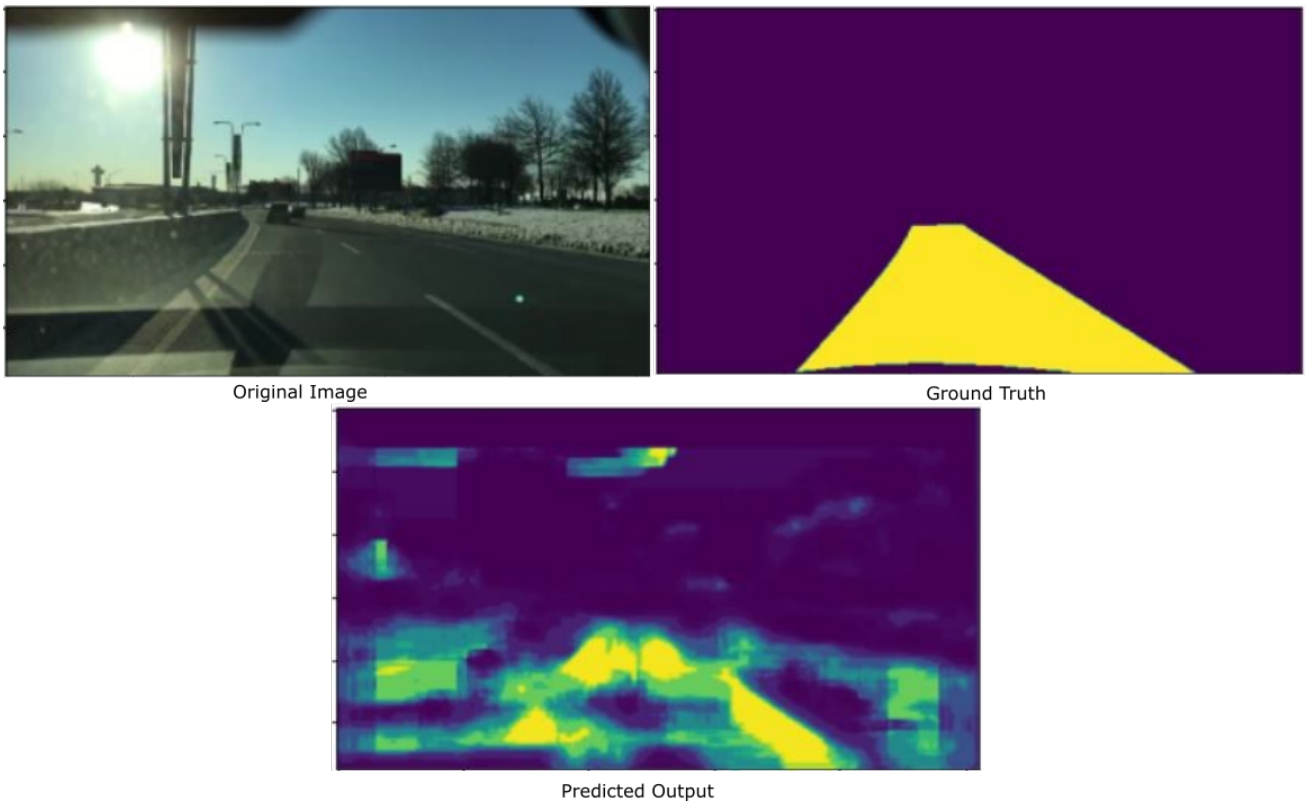

Figure 4: Results from Non-Optimized Model

3.5 Hyperparameter Tuning

To improve the model results and better fit to our data hyperparameter tuning technique is used.

We considered these hyperparameters: Number of Epochs, Image Size, Batch Size, Kernel Size, Learning Rate, Layers

Our selection of hyperparameters was based on the two factors, one computational power available to us. As we were using the palmetto cluster, each time a high specced node wasn't available, so we varied the batch size accordingly.

Second, by studying the output metrics from each change in hyperparameter tuning step. Such as, while increasing the number of epochs, after 5-6 epochs till the 10th epoch, the values of validation loss, accuracy and mean IoU settled to a value. This is evident from Fig. 3. Results from hyperparameter tuning listed on Table 3.

Trial No.	Epoch	Image Resize Factor	Batch Size	Learning Rate	mIOU	Training Time (s)	Validation Loss
1	10	0.6	8	0.001	0.41	610	0.118
2	10	0.6	8	0.0001	0.43	1170	0.157
3	10	0.6	8	0.1	0.44	1165	0.121
4	10	0.6	16	0.001	0.44	2152	0.068
5	**10**	**0.6**	**32**	**0.001**	**0.442**	**1230**	**0.0705**
6	30	0.6	32	0.0001	0.44	1490	0.0713

Table 3: Hyperparameters tuned. Highlighted ones indicate optimized model parameters

It is evident from Table 3, tuning hyperparameters did not make much of a difference to the mean IoU. Hence we used the "Visualization of output label" as our evaluation metrics for tuning the model.

Even though the optimized model has the same mean IoU but the validation loss is lowest amongst all. The number of layers played a significant role in the resultant output predictions of the model. Since the predicted label of the model as seen in Fig 4. was unable to learn all the features appropriately. Adding more convolutional layers and subsequent convolutional layers was the next logical step

3.6 Optimum Model

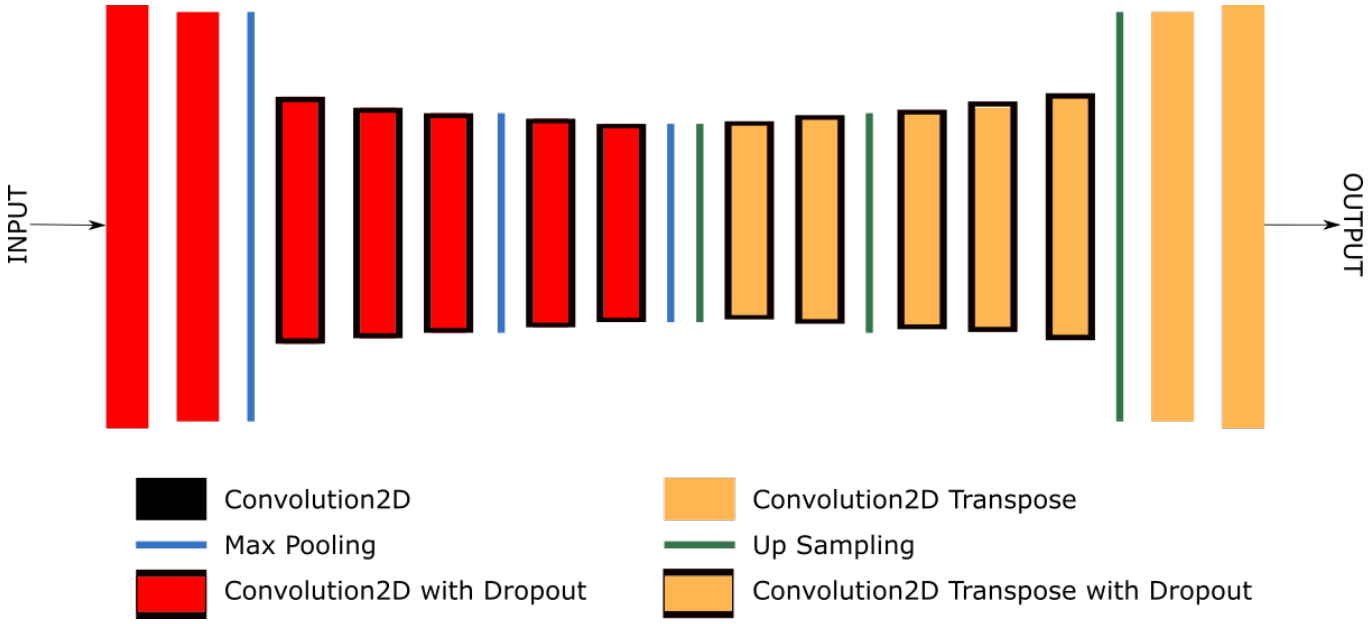

Figure 5: Optimized Network Architecture

Layers Description: The final model used for training has in total 20 layers consisting of 7 Convolution Layers, 7 Deconvolution layers, 3 Max pooling layers, 10 Drop out Layers, 3 Up-sampling layers. 2 convolution layers followed by a max pooling layer, then 3 convolution layers followed by a max pooling layer, again 2 convolution layers followed by a max pooling layer. Max-pooling is done here by applying a max-filter of a stride (1,1). This reduces the image resolution to (50, 92). This provides basic translational invariance to the internal representation.

After this we have used an Up-Sampling layer. Up-sampling reduces the computations in each layer by keeping the dimensions input as before. This is followed by 2 Deconvolutional layers, then again one Up-Sampling layer followed by 3 Deconvolutional layers. Then again one Up-Sampling layer followed by 2 Final Deconvolutional Layers. Each layer used a ReLU activation function. To help avoid overfitting, after each layer a DropOut of 0.2 has been added. Before feeding to the network the data has been shuffled to reduce the overfitting.

Hyperparameter	**Values**
Image Size	60% of Original
Training Data	12504
Testing Data	3126
Batch Size	32
Epoch	10
Optimizer	ADAM (LR = 0.001)
Loss Function	Mean Squared Error
Activation Function	ReLU

Table 4: Optimized Model Parameters

Model Training: The training of the model with 60% image size and using mean squared error loss for 10 epochs is shown in Fig. 3 We see a constant mean IoU of 0.4415.

Optimum Model Results: The predicted results of the optimum model can be seen in the figures below. On comparing the predicted label with ground truth, the feature learning is almost similar to the ground truth label wherein the outlines of the drivable area are properly segmented. This can also be observed in the video image frame captured in Fig. 4.

```
Train on 12504 samples
Epoch 1/10
12504/12504 [==============================] - 150s 12ms/sample - loss: 0.1172 - mean_io_u: 0.4415
Epoch 2/10
12504/12504 [==============================] - 143s 11ms/sample - loss: 0.1170 - mean_io_u: 0.4415
Epoch 3/10
12504/12504 [==============================] - 144s 11ms/sample - loss: 0.1170 - mean_io_u: 0.4415
Epoch 4/10
12504/12504 [==============================] - 143s 11ms/sample - loss: 0.1170 - mean_io_u: 0.4415
Epoch 5/10
12504/12504 [==============================] - 144s 11ms/sample - loss: 0.1170 - mean_io_u: 0.4415
Epoch 6/10
12504/12504 [==============================] - 143s 11ms/sample - loss: 0.1170 - mean_io_u: 0.4415
Epoch 7/10
12504/12504 [==============================] - 143s 11ms/sample - loss: 0.1170 - mean_io_u: 0.4415
Epoch 8/10
12504/12504 [==============================] - 144s 11ms/sample - loss: 0.1170 - mean_io_u: 0.4415
Epoch 9/10
12504/12504 [==============================] - 144s 11ms/sample - loss: 0.1170 - mean_io_u: 0.4415
Epoch 10/10
12504/12504 [==============================] - 144s 11ms/sample - loss: 0.1170 - mean_io_u: 0.4415
```

Figure 3: Runtime output from Optimal Model Training

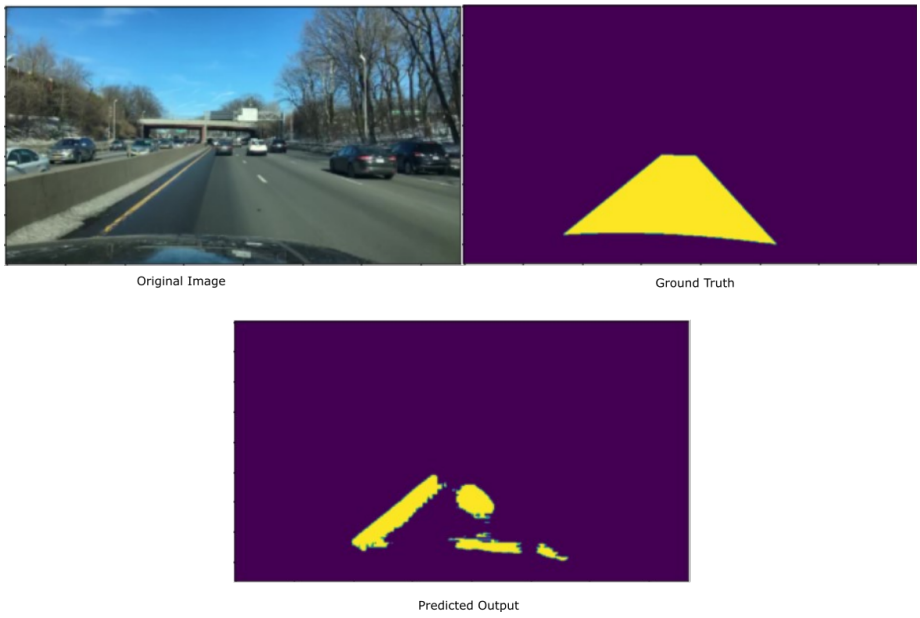

Figure 4: Results from Optimized Model

4 Conclusion

Here we presented a simple CNN that can process an image taken in a worse weather environment and predict the drivable area in the image. Our simple hyperparameter tuning was limited by the hardware limitations in some cases, we were able to modify our simple model to a better accuracy with less number of iterations. We tested on the curated dataset from the state-of-the-art BDD100K dataset. In the dataset we selected only worse weather conditions for our case and were able to demonstrate that the algorithm is robust enough to give consistent results in varying weather conditions.

5 Future Scope

- As observed from the results of the initial model and final optimum model, the feature learning of the images was improved. However, the improvement is not sufficient which can be observed in the video frame upon deploying this model for foggy weather test cases. In order to improve the feature learning additional

convolutional layering followed by deconvolution layering can be implemented. Fine tuning of the model is required to reduce the disparity between predicted and ground truth.

- The model needs more tuning of hyperparameters for optimum performance, for example the model can be trained for more epochs with a dual GPU with better learning rate. Since the loss of the model reduces minutely throughout the 10 epochs, if the learning rate is increased while allowing more repetitions of training(epochs) a better model output can be achieved.
- The time taken for model training is one major bottleneck even for small image sizes, we were training the model on single GPU however using dual GPU power the training time can be reduced for big data size and thereby using original image size of 720 x 1280 pixels can be then used as an input to the model. Thus a more complex model with higher dimensions of image, a better feature learning can be done.
- We used the network to predict lanes on a video. That way using the same network and adding a Recurrent Neural Network to make use of the temporal features would yield better results.

Author Contributions: All authors contributed equally in researching, implementation and writing the report.

Acknowledgements: We thank Rahul Rai PhD, and Shengli Xu for their valuable guidance. Clemson University is acknowledged for generous allotment of compute time on Palmetto cluster.

Code Availability: Implementation and a Demo video is available at `https://github.com/ppswaroopa/Lane-Detection-In-Foggy-Weather`

References

[1] Dinesh Bharadia. Researchers working to improve autonomous vehicle driving vision in the rain - ai trends, 2021.

[2] John P. Desmond. Waypoint - the official waymo blog: A fog blog: Understanding a challenge inherent to driving in san francisco, 2021.

[3] Jung-Hwan Kim, Sun-Kyu Kim, Sang-Hyuk Lee, Tae-Min Lee, and Joonhong Lim. Lane recognition algorithm using lane shape and color features for vehicle black box. In *2018 International Conference on Electronics, Information, and Communication (ICEIC)*. IEEE, 1 2018.

[4] Lizhe Liu, Xiaohao Chen, Siyu Zhu, and Ping Tan. Condlanenet: a top-to-down lane detection framework based on conditional convolution. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 10 2021.

[5] Xiaolong Liu and Zhidong Deng. Segmentation of drivable road using deep fully convolutional residual network with pyramid pooling. *Cognitive Computation*, 10(2):272–281, 11 2017.

[6] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. 11 2014.

[7] Nan Ma, Guilin Pang, Xiaojun Shi, and Yun Zhai. An all-weather lane detection system based on simulation interaction platform. *IEEE Access*, 8:46121–46130, 2020.

[8] manoshape Michael Virgo. Mlnd-capstone, 2020.

[9] Muhammad Jefri Muril, Nor Hidayati Abdul Aziz, Hadhrami Ab. Ghani, and Nor Azlina Ab Aziz. A review on deep learning and nondeep learning approach for lane detection system. In *2020 IEEE 8th Conference on Systems, Process and Control (ICSPC)*. IEEE, 12 2020.

[10] D. Pomerleau. RALPH: rapidly adapting lateral position handler. In *Proceedings of the Intelligent Vehicles '95. Symposium*. IEEE, None.

[11] xenons4u.co.uk. Mercedes-benz active lane keeping assist inoperative warning message, 2019.

[12] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. BDD100K: A diverse driving dataset for heterogeneous multitask learning. 05 2018.

[13] M. Mahmud Yusuf, Tajbia Karim, and A. F. M. Saifuddin Saif. A robust method for lane detection under adverse weather and illumination conditions using convolutional neural network. In *Proceedings of the International Conference on Computing Advancements*. ACM, 1 2020.