TOPIC:

ROAD – LANE SEGMENTATION

USING PYTHON

BY

GAURAV PATIL (gpati4@uis.edu)

ABSTRACT

Lane detection on the road is a critical component in the development of expert driver assistance systems and self-driving cars. Deep learning approaches have demonstrated exceptional performance in road lane recognition in recent years, owing to their capacity to learn complicated features and patterns from photos. A study on road lane recognition using deep learning algorithms is presented in this project. In this project we identified the lane marks on the road by masking the images. For this project we used CNN and plotted the values for overall validation loss using 3 different optimizers – RMSprop , Adam and Nadam. RMSprop had the lowest validation loss as when compared to that of Adam and Nadam. But the validation loss increases in case of RMSprop instead of decreasing when compared to Adam and Nadam. In the project the loss value was much higher but when the model was tuned the validation loss dropped significantly expect for RMSprop.

CONTENTS

LIST OF FIGURES

CHAPTER 1

PROBLEM DEFINITION AND GOALS

The problem we aim to address in this project is the detection of road lane markings using deep learning technique i.e., CNN (Convolution Neural Network). Road lane markings play a critical role in ensuring road safety, guiding drivers, and enabling autonomous vehicles to navigate roads. However, detecting road lanes can be challenging due to factors such as poor lighting conditions, occlusions, and variations in lane markings. Hence, for this we are masking the images of the road lane so that we overcome the challenges. The dataset we used for the project included the images (.jpg format) along with the grayscale images (also in .jpg format). This dataset contained 6000 images of the lane. We obtained this dataset from the web. From these images we will mask the images such that only the lane marking are highlighted. We will the compare them to the readily available masked images and the desired output by the end we are expecting is to have a proper masked image such that only the lane is properly highlighted.

CHAPTER 2

RELATED WORKS

- **The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation by** Simon Jegou, Michal Drozdzal, David Vazquez, Adriana Romero, Yoshua Bengio, Montreal Institute for Learning Algorithms.

  In this paper, they have extended DenseNets to deal with the problem of semantic segmentation. They achieve state-of-the-art results on urban scene benchmark datasets such as CamVid and Gatech, without any further post-processing module nor pretraining. Moreover, due to smart construction of the model, their approach has much less parameters than currently published best entries for the datasets. They used the FC-DenseNet103 model pretrained on CamVid, removed the softmax layer, and finetuned it for 10 epochs with crops of 224 × 224 and batch size 5(In our project we have used a image size of 180 X 180). Given the high redundancy in Gatech frames, we used only one out of 10 frames to train the model and tested it on all full resolution test set frames.

- **CNN based lane detection with instance segmentation in edge-cloud computing by** Wei Wang, Hui Lin & Junshu Wang

  In this paper, the lane detection algorithm is carefully studied. The lane detection method based on traditional image processing and the lane detection method based on deep learning are analyzed and compared to solve the problem of lane detection under complex conditions such as shadows and obstacles. In order to improve the efficiency of central computing, a two-branch training network and a customized training network based on semantic segmentation are proposed, and the combination of vehicle edge calculation and actual engineering can improve the effect of lane line detection network. In the inverse perspective transformation, the use of a fixed transformation matrix will cause errors when ground changes, which will cause the vanishing point projected to infinity to move up or down. We trained a neural network with a custom loss function that can dynamically predict the parameter values of the transformable matrix. The transformed lane points are fitted with second or third-order polynomials. The prediction is based on the input image, allowing the network to adjust the projection parameters when the ground plane changes, making the model excellent stickiness. The final tests show that the model in our paper has better performance in scenarios such as insufficient lighting and lane line degradation.

- **Real Time Road Lane Detection Using Deep Convolutional Neural Network by** R Shreyas, Sai Karthik P K, R Ajay, Prof. Karthik S A

  It is evident that recent advances have been made in the detection of road lanes on driving scenes. Even though there are several methods that

achieved significant advancements using high precision and efficient methodologies, there are still many challenges pertaining to extreme driving conditions that have to be addressed. To overcome all these challenges this publication has proposed an optimized and hybrid combination of Deep CNN and RNN. The reason they choose CNN is that it can extract the spatial from the data using kernels, which other networks are not capable of. The proposed method uses a combination of DCNN and RNN to predict road lanes using a continuous sequence of frames as an input. Through vigorous experimentation and testing, the model has achieved a significant accuracy of 96% and a precision of 0.964. They have also demonstrated the step by step inner workings of the model in the output, for a better visual understanding of the lane detection process by the model.

- **Road Lane Detection using Convolutional Neural Network by**
  Farjana Farvin S , Sowndarya S V
  Anjalai Ammal Mahalingam Engineering College, Thiruvarur, India

  To handle the problem of lane detection in difficult conditions such as shadows and obstructions, the lane detection methods based on classical image processing and the lane detection approach based on deep learning are analyzed and contrasted in this publication. A two-branch training network and a tailored training network based on semantic segmentation are proposed to improve the efficiency of central computing, and the combination of vehicle edge computation and actual engineering can improve the effect of lane line detection networks. When the ground moves in the inverse perspective transformation, the usage of a fixed transformation matrix causes mistakes, causing the vanishing point projected to infinity to move up or down. The

model's stickiness is enhanced by the fact that the prediction is based on the input image, which allows the network to modify the projection parameters when the ground plane changes. The results of the final testing suggest that the model resented in their research performs better in conditions such as poor lighting and lane line erosion.

CHAPTER 3

DATA EXPLORATION AND PREPROCESSING

The data which we used in the project were all images in .jpg format. First we extracted the zip folder and found 2 directories in the folder – Train and Masked. For exploring the data we just loaded the image from the Train folder and at the same time we also loaded the image from the Masked folder. Then we got a list of number of images in each folder and found that they had 6000 images in the Train folder and 6000 images in the Masked folder. We then normalized the image based on its type. For that we have written a function. If the argument is mask then the function subtracts 1 from input image so that they start from 0 instead of 1. If the type argument is not mask then the function normalizes the images by dividing its pixels by 255.

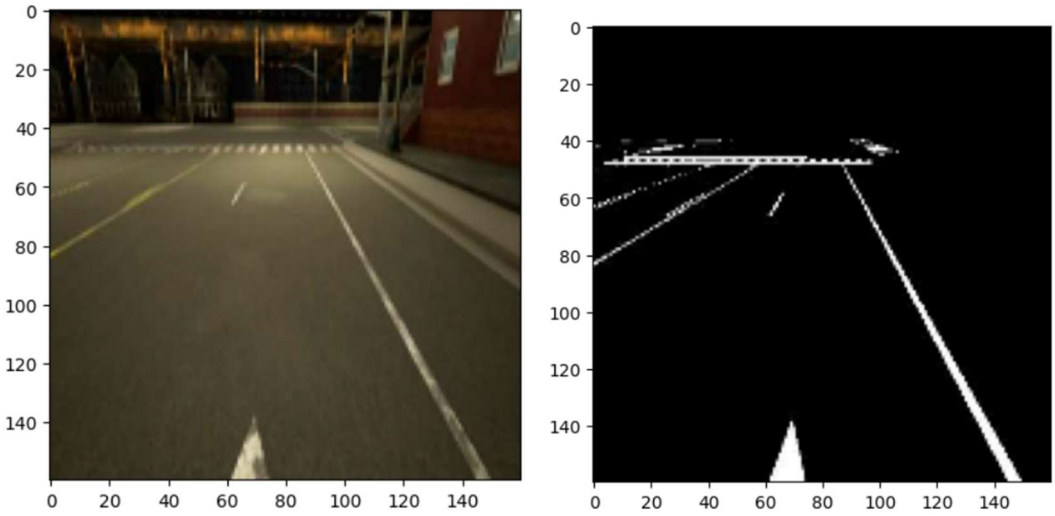

Fig 3.1 Masked Image

This is done to scale the image at the range of 0 to1. This is an important step as it improves the performance of the model during the training. We used globe() to get a list of images path names. Also, we used the tf.data.Dataset.from_tensor_slices() to create a dataset containing tuples. We have also used map() to each element in the dataset and this function returns a new dataset containing the results. "DenseNet121" architecture from the Keras Applications module, which is a pre-trained deep learning model that is trained on the ImageNet dataset for image classification. Next, the code sets the "trainable" attribute of the convolutional base to "False" to freeze the pre-trained weights of the model. By freezing the weights of the convolutional base, the model will not modify the weights during training, and only the new layers added on top of the convolutional base will be trained.

CHAPTER 4

DATA ANALYSIS

For the project we have used Convolutional Neural Network. We have used the DenseNet121 Architecture in this Neural Network. The DenseNet121 architecture has a total of 121 layers, including convolutional layers, dense layers, and other types of layers. However, since we have set include_top=False, the fully connected layers at the end of the network have been excluded, and only the convolutional base of the network is being used. Therefore, the number of layers and neurons used in your model will depend on the specific configuration of the conv_base model. We can see the exact number of layers and their shapes by calling conv_base.summary() as shown in the code snippet. In the Input Layer we have given the shape as (160,160,3) for the neuron and the expected output we got was (None,160,160,3). The next layer was zero padding 2D which also had the same shape as the Input Layer 1. The Conv2D, BatchNormalization, Activation has the shape (None,80,80,64). Similarly other layers have their expected output shape and the parameters which are displayed by the conv_base.summary(). The dense connection provides a number of advantages. DenseNet121 provides various benefits over other common CNN designs like as VGG, ResNet, and Inception. For starters, its dense connectedness

promotes greater feature reuse and aids in the alleviation of the vanishing gradient problem, resulting in improved gradient flow and simpler training of deeper networks. Second, compared to other deep learning architectures, it has fewer parameters, making it more computationally efficient and easier to train on smaller datasets. Finally, DenseNet121 has demonstrated that it can achieve comparable or better accuracy than other architectures while employing fewer layers and parameters. Since our project doesn't have much parameters to compute we have elected DenseNet121 architecture. First, we created a variable "num_classes" with the value of 3, which represents the number of classes for lane segmentation. Then, we added a new convolutional layer to the output of the pre-trained DenseNet121 model using the "Conv2D" layer from Keras. This layer has a kernel size of (1,1), which means it will apply a 1x1 convolution to the output feature maps of the pre-trained model. The "padding" parameter is set to "same" to ensure that the output feature maps have the same size as the input feature maps. The "activation" parameter is set to "relu" to introduce non-linearity in the output feature maps. Next, we added "Conv2DTranspose" layer to the model. This layer is used for upsampling the output feature maps from the previous convolutional layer to the original input image size. The "kernel_size" parameter is set to 64 to define the size of the upsampling kernel. The "strides" parameter is set to 32 to define the upsampling factor. The "padding" parameter is set to "same" to ensure that the output feature maps have the same size as the input feature maps. The "activation" parameter is set to "softmax" to ensure that the output feature maps represent probability distributions over the lane classes. Finally, a new Keras model will be instantiated with the inputs from the pre-trained DenseNet121 model and the output from the "Conv2DTranspose" layer. The model is trained the previously defined fit() method in Keras. The model is compiled with the RMSprop optimizer and categorical crossentropy loss function. The training data is passed to the train_ds variable, which

should be a dataset object containing the training data. The number of epochs is set to 100, which means that the model will be trained for 100 iterations over the entire training dataset. The validation data is passed to the val_ds variable, which should be a dataset object containing the validation data. The model will be evaluated on this data after each epoch. The callbacks parameter is a list of Keras callbacks that will be used during training. These callbacks include:

ModelCheckpoint: This callback will save the model weights to a file whenever the validation loss improves.

EarlyStopping: This callback will stop training if the validation loss does not improve for a specified number of epochs (patience).

ReduceLROnPlateau: This callback will reduce the learning rate if the validation loss does not improve for a specified number of epochs (patience).

Finally, the fit() method is called on the model object with the specified training and validation datasets, as well as the callbacks. The history variable will contain the training and validation loss and accuracy values for each epoch.

We expect to get a loss value of around 0.25 with the validation loss of around 0.34 as the dataset is seems to be quite decent.

CHAPTER 5

EVALUATING, TUNING AND IMPROVING MODEL

From the results of the training, it is quite clear that the model is overfitting on a larger extent. The loss value using the RMSprop optimizer was 38.026 during the 6th iteration and in the end the validation loss for the same turned out to be 40.826 which is comparatively high.

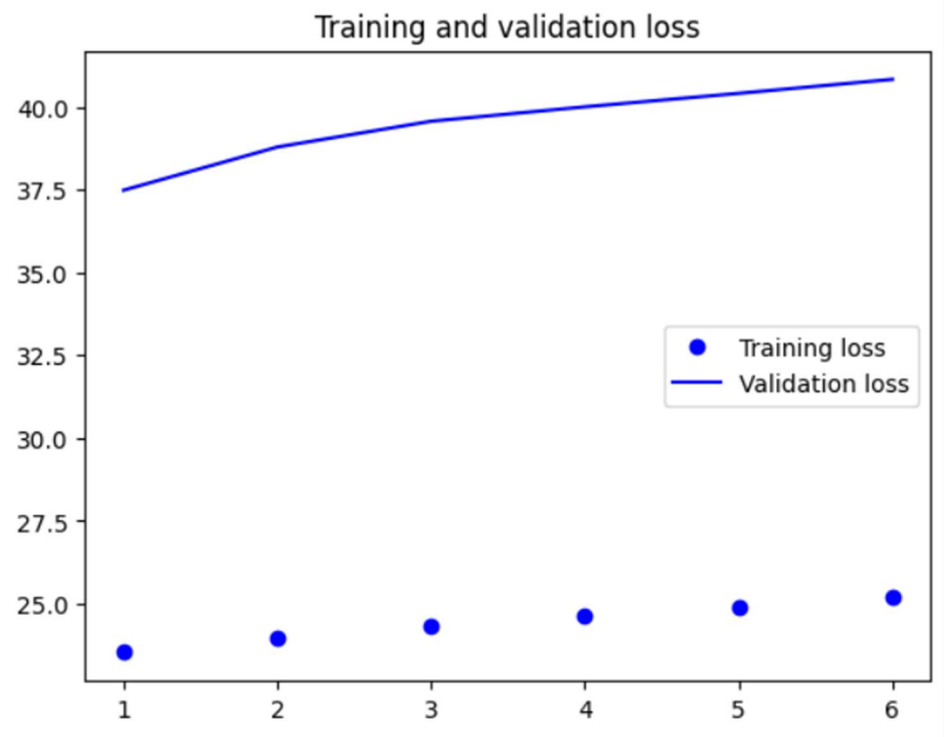

Fig 5.1 Training and validation loss

Hence, we tried to fine-tune the model using a lower learning rate 1e-4 and the previous one which we used was 1e-3. The images plotted were as follows:

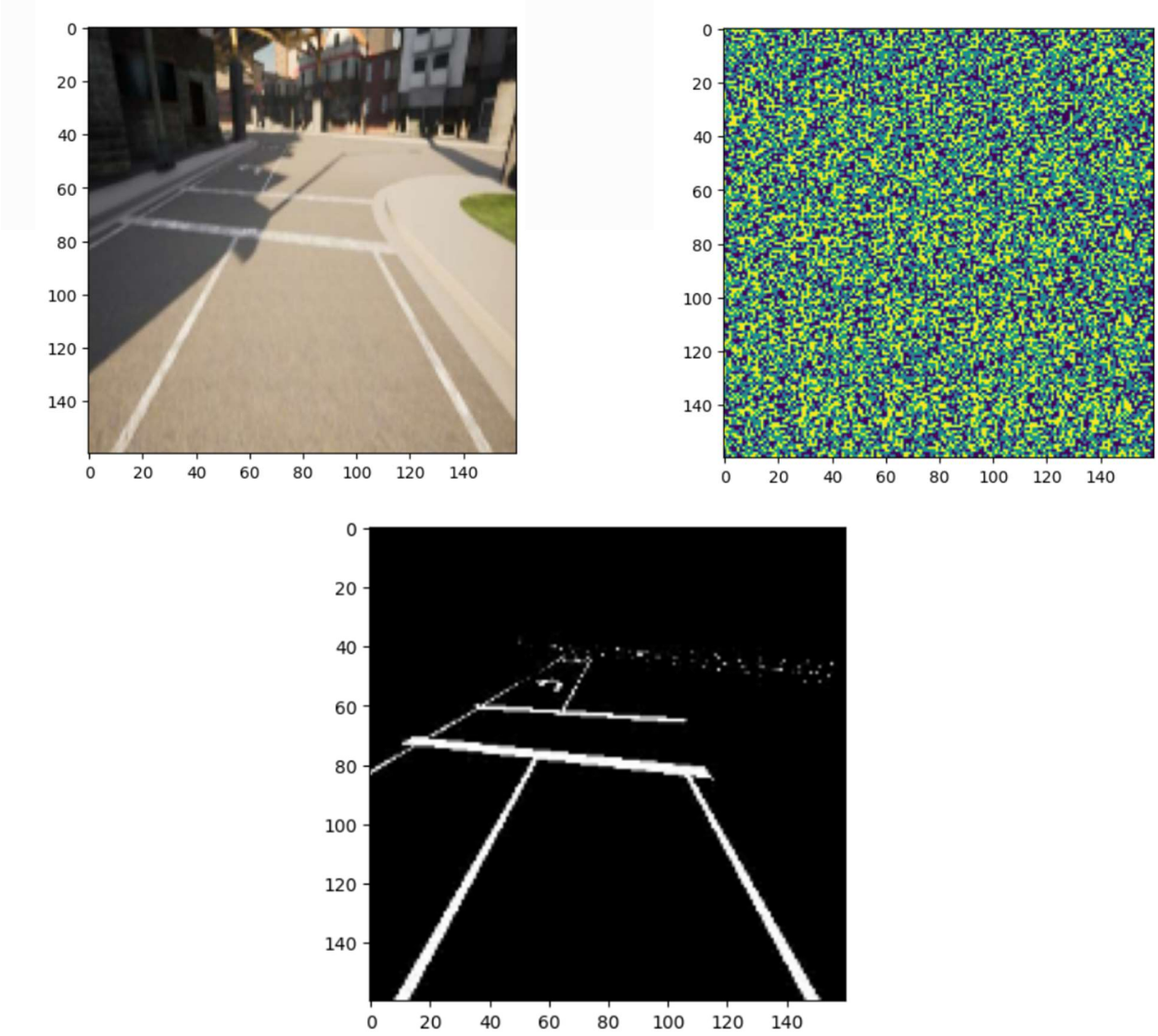

Fig 5.2 Masks

The images shown above indicate how well the lane is detected and masked. For each image in the batch, the model's prediction is a pixel-wise probability distribution over three classes. The code converts this output to a single-channel mask by taking the class with the maximum predicted probability using np.argmax. For the first five images in the batch, the code displays the original image, the predicted mask, and the true mask side by side using tf.keras.utils.array_to_img and matplotlib.pyplot.imshow. Next unfreeze the weights of the convolutional base of the previously trained model. It sets trainable attribute to true for all layers except the last 9 layers. We define the optimizer for the model, which is RMSprop with a learning rate of 1e-4. Then, we will call the compile() method on the model with this optimizer and the loss function set to categorical cross-entropy.

The code shows the training process of a neural network for 100 epochs with a batch size of 150. During each epoch, the model is trained on the training dataset and evaluated on a separate validation dataset. The loss values for both training and validation sets are recorded after each epoch.

Based on the results provided, we can see that the training loss (loss) and validation loss (val_loss) are both high and increasing over time. This indicates that the model is not learning to fit the training data well and is overfitting to noise in the data. The learning rate (lr) is set to a small value of 1.0000e-04.

To improve the performance of the model, it may be necessary to adjust the learning rate or other hyperparameters, such as the regularization strength or the number of layers.

Another approach which we can use is adaptive learning rate optimization algorithm, such as Adam which automatically adjust the learning rate during training based on the gradients and past updates. These algorithms can help speed up the convergence and improve the performance of the model. Even if we try setting the learning rate high for this dataset it turns out that the loss only increases exponentially.

So now we tried using the Adam optimizer to see if the loss actually decreases or not. The results which we got were quite interesting. The validation loss for the model using the Adam optimizer was 74.615 which was quite high when compared to the validation of the RMS prop optimizer.

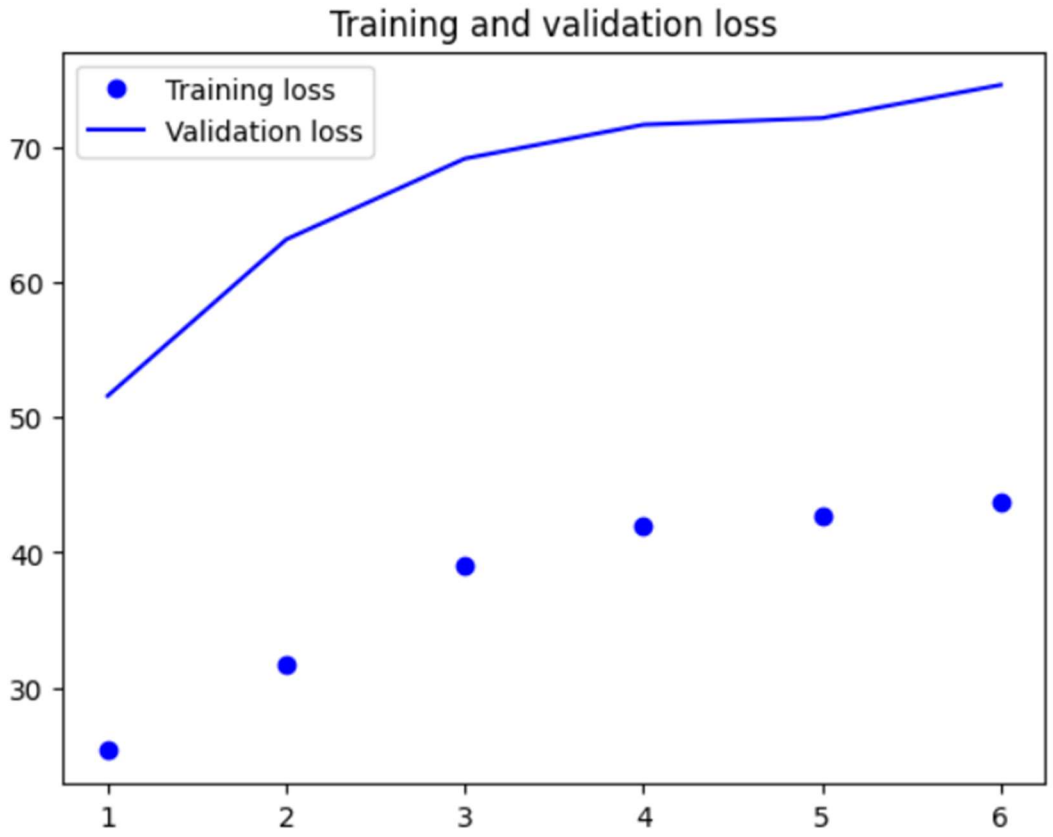

Fig. 5.3 Training and validation loss for Adam optimizer

After fine Tuning the model using adam optimizer we got a much lower validation loss of about 49.69. Thus, we can see a significant improvement in the model in terms of reducing the validation loss. But when compared to the RMSprop optimizer this validation loss of 49.69 is still much higher.

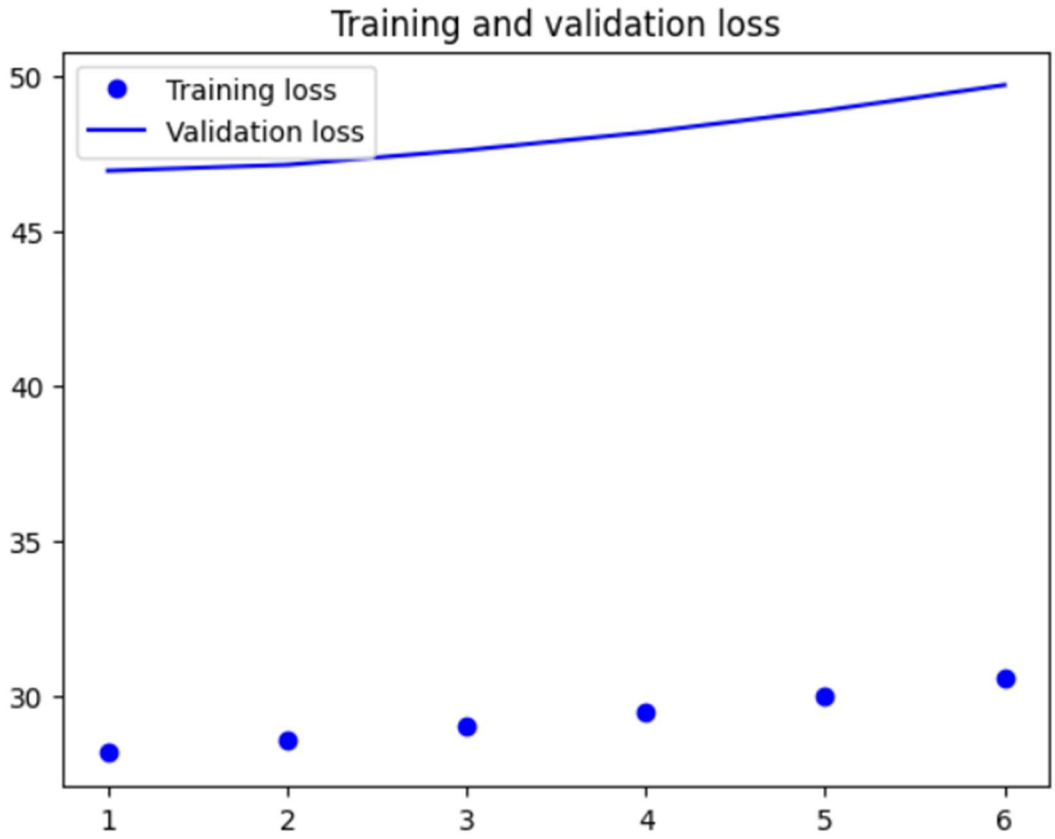

Fig. 5.4 Training and Validation Loss for Adam Optimizer after fine tuning

Next we tried using the Nadam optimizer to see if we get any improvement in the model or not. The validation loss for the Nadam optimizer was 78.72 and this was prior to the fine tuning of the model.

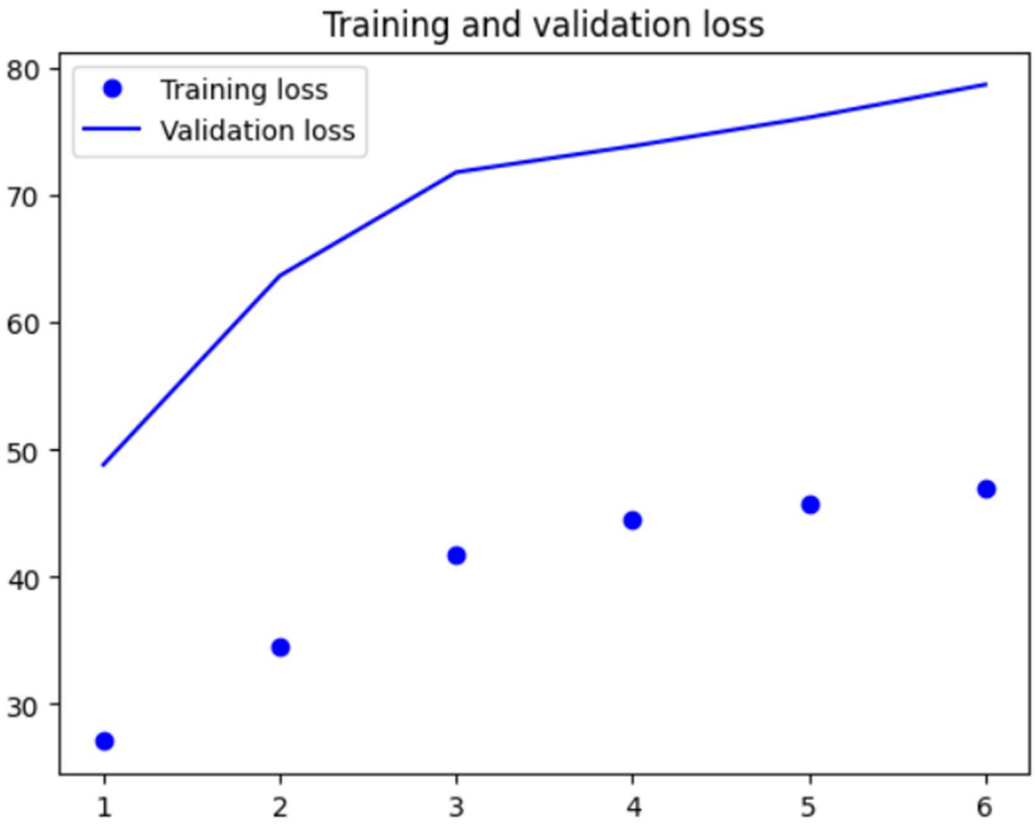

Fig 5.5 Training and Validation loss for Nadam Optimizer

Next we tried fine-tuning the model based on the same optimizer and the a reduced learning rate of 1e-4. The validation loss which we got was 52.928 which was also lower than the validation loss prior to the tuning of the model.

From the results it is clear that the validation loss is less only when we use the RMSprop optimizer but the validation loss only increases after each epoch. But in the case of the Adam and Nadam optimizer the validation loss decreases after the model is tuned. In this way we tried improving the model by decreasing the validation loss by changing the optimizers.

The model was overfitting in each and every method we did for decreasing the validation loss but it didn't seem to fit properly.

CHAPTER 6

CONCLUSION

From the above results we have got a clear idea that the model did not fit properly due to the noise in the dataset. The validation loss was significantly high before tuning the model. Lets compare the tuning results of all the 3 optimizers.

For RMSprop optimizer, the training loss and validation loss decrease slowly over the first 6 epochs. This could suggest that the learning rate is too low and the model is not converging fast enough.

For Adam optimizer., the training loss and validation loss start at a higher value and increase over the first 6 epochs. This could suggest that the learning rate is too high and the model is overshooting the optimal weights.

Comparing the two sets of results, it seems that the first set of results performs better than the second set. However, it is important to note that the choice of learning rate also depends on other factors such as the specific dataset and architecture being used.

Lets compare the result of Adam and Nadam optimizer. It looks like the Nadam optimizer had a higher loss than the Adam optimizer for each epoch, both on the training and validation set. This suggests that the Adam optimizer performed better than the Nadam optimizer.

Based on the given output, it seems like the loss values are increasing with each epoch in all three cases, which is not a good sign. This could be an indication of a high learning rate or poor model architecture. To improve the results, we can try adjusting the learning rate or modifying the model architecture.

We can try decreasing the learning rate from the current 1e-4 to 1e-5 or 1e-6. We can also use a learning rate scheduler to gradually decrease the learning rate over time.

We can also try modifying the model architecture by adding more layers, adjusting the number of filters in the convolutional layers, or increasing the number of neurons in the dense layers. Additionally, we can experiment with different loss functions or regularization techniques to improve the performance of the model.

However, it's important to note that the comparison is only based on the loss metric, and there could be other factors to consider as well, such as accuracy, precision, recall, etc.

There are not many works on this current dataset used in the project so it is difficult to compare the results.

	Before Tuning		After Tuning	
Optimizer	Loss	Validation loss	Loss	Validation Loss
RMSprop	24.06	38.02	25.18	40.82
Adam	43.72	74.61	30.55	49.69
Nadam	46.98	78.72	32.68	52.92

CHAPTER 7

REFERENCES

- https://ieeexplore.ieee.org/document/8014890
- J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.
- L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In International Conference of Learning Representations (ICLR), 2015.
- CNN based lane detection with instance segmentation in edge-cloud computing by Wei Wang, Hui Lin & Junshu Wang
- Real Time Road Lane Detection Using Deep Convolutional Neural Network by R Shreyas, Sai Karthik P K, R Ajay, Prof. Karthik S A

CHAPTER 8

DATASET LINK

https://drive.google.com/file/d/11XkrZDH_pqc0SFRJufETLipXkVEmUP_E/view?usp=sharing

https://drive.google.com/file/d/1LgOZlWd5Qi09Bun3vRmbzRwFXADOXUDE/view?usp=share_link

CHAPTER 9

SOURCE CODE COLAB LINK

https://colab.research.google.com/drive/19yMrlOmt5503aFfb9ZIQ2Yy2C81YC3cy?usp=sharing