MA-INF 4308 - Lab Vision Systems -Video Segmentation and Understanding of Automotive Scenes

Abhishek S Pillai(50010996) and Areesha Asif(50091632)

Master of Science Computer Science (M. Sc.), Institute of Computer Science,University of Bonn,Germany

Abstract. Advancements in computer vision tasks have been pivotal in facilitating intelligent systems to perceive and interpret the world around them. Our project presents an unified approach to three critical aspects of visual understanding: semantic segmentation, depth estimation, and camera pose estimation. By designing a multi-task learning framework that leverages *Convolutional and Recurrent neural networks*, we aim to create a cohesive system capable of performing these tasks simultaneously, with improved efficiency and accuracy. We have done rigorous training and validation procedures on three models developed with dataset variants also utilizing state-of-the-art loss functions to ensure effective learning.The project's outcomes contribute valuable insights into the domain of Vision systems and machine learning, showcasing the potential of multi-task learning systems in achieving nuanced visual understanding.

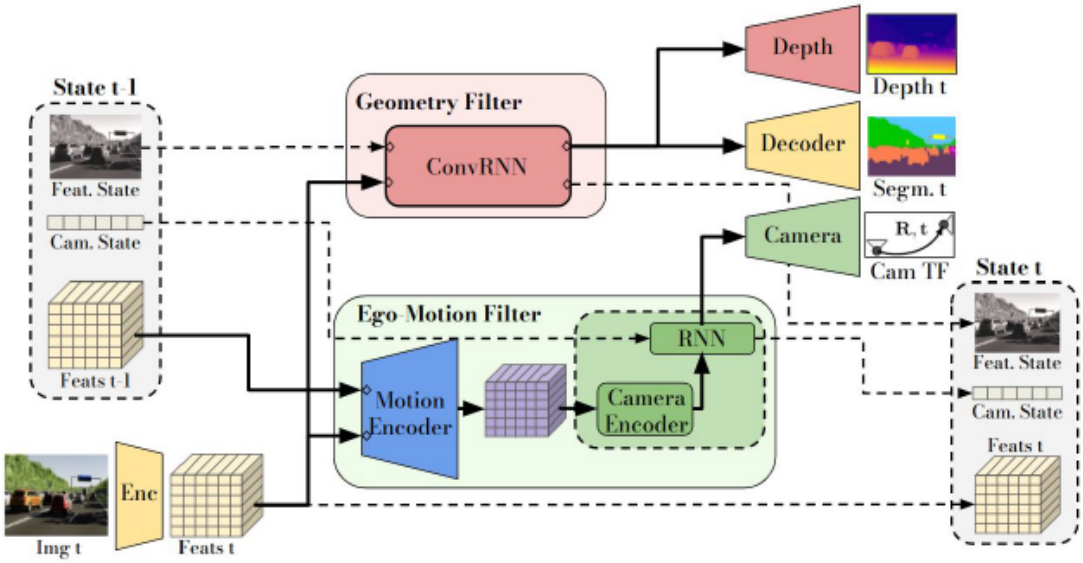
1 Introduction

In an era where machine perception is paramount, the quest for creating models that emulate the human eye's discernment has led to remarkable breakthroughs.With the advancement of autonomous driving technologies and Robotics,the ability to accurately interpret and understand automotive scenes through video segmentation has become paramount.

The model presented herein represents a pioneering foray into the domain of integrated visual understanding.The model is a strategic amalgamation of convolutional layers and recurrent neural dynamics, encapsulated in a Convolutional Recurrent Neural Network (ConvRNN).It captures the essence of time, infusing the model's spatial acuity with a temporal dimension.Parallelly, the Ego-Motion Filter, consisting of a Motion Encoder and a Camera Encoder, together with a Recurrent Neural Network (RNN), crafts a nuanced understanding of motion and transformation .This dual-layered architecture juxtaposing geometry and ego-motion—forms a complex yet harmonious interpretation in our model.

In this report we start with the basic method of our architecture giving a brief overview of our model.In the experimental section we delve deep into the dataset, the different models implemented mainly categorized into Geometric filter and Ego-Motion filter.In the training section we explain how our code was implemented, its structure, the parameters and loss calculation formulas used.Later in the results section, we present visualization and output values we got in our training.Finally in the last part we discuss the conclusion.

2 Method



In this section we will outline the basic structure and functioning our model architecture.Central to our model is the Geometric filter and Ego Motion Filter.

- **Geometric Filter**-capturing and maintaining the structural and spatial consistency of the scene over time.The ConvRNN cell in the Geometric Filter combines the current frame's features with the accumulated knowledge from previous frames to make informed predictions about the scene geometry, even in cases of partial occlusions or motion blur.It produces two outputs—semantic segmentation maps and depth maps—which are fundamental for understanding the scene both categorically and spatially.
- **Ego Motion Filter**-Basically it focuses on the camera's movement and its implications for scene interpretation which is particularly important for applications where the observer or robot is in motion (1).Thus through camera modeling it helps in understanding the scene changes, information about the observer's trajectory and orientation changes and also helping in disambiguating object motion from camera motion. These processing is also done with the help of a RNN cell.

Overall it employs a Recurrent Semantic Segmentation Model to enforce the consistency of segmentation outputs over time. It does so by taking the encoded features and passing them through the above filter model that progressively refines the segmentation predictions for each frame.

3 Experimental Details

Dataset- We are using the CARLA Dataset for autonomous driving generated with the CARLA simulator. It contains approx. 12000 sequences 20 frames of size (512x1024). For Training, we are using Towns 01, 03, 04, 05, 06, 07 . For the Validation part we are using Town 02 and for Testing Town 10. Because of the ts simulated nature of the dataset it provides labels for Camera poses, intrinsic Semantic segmentation, and Depth estimation. First we tried to inspect the dataset's directories and files present in those directories. After complete understanding. We tried to visualise some images from dataset to see what kind of images are there what is the shapes of these images etc. There are other files also present in dataset instead of images which includes meta.pkl files and optical flow files. We tried to print the content of these files as well to understand what kind of data is in these files. Then after complete understand of dataset and making a road map of how we can use given files in different components of our architecture,

we proceeded to write code logic to relocate the dataset according to our requirement of only 6 frames. After complete relocation we tried to corrupt the BASE CARLA dataset using different corruption techniques. So this is how we prepared our dataset.

We have used 2 variants of the dataset. One is the *Base CARLA dataset* which includes a sequence of 6 images from the dataset.

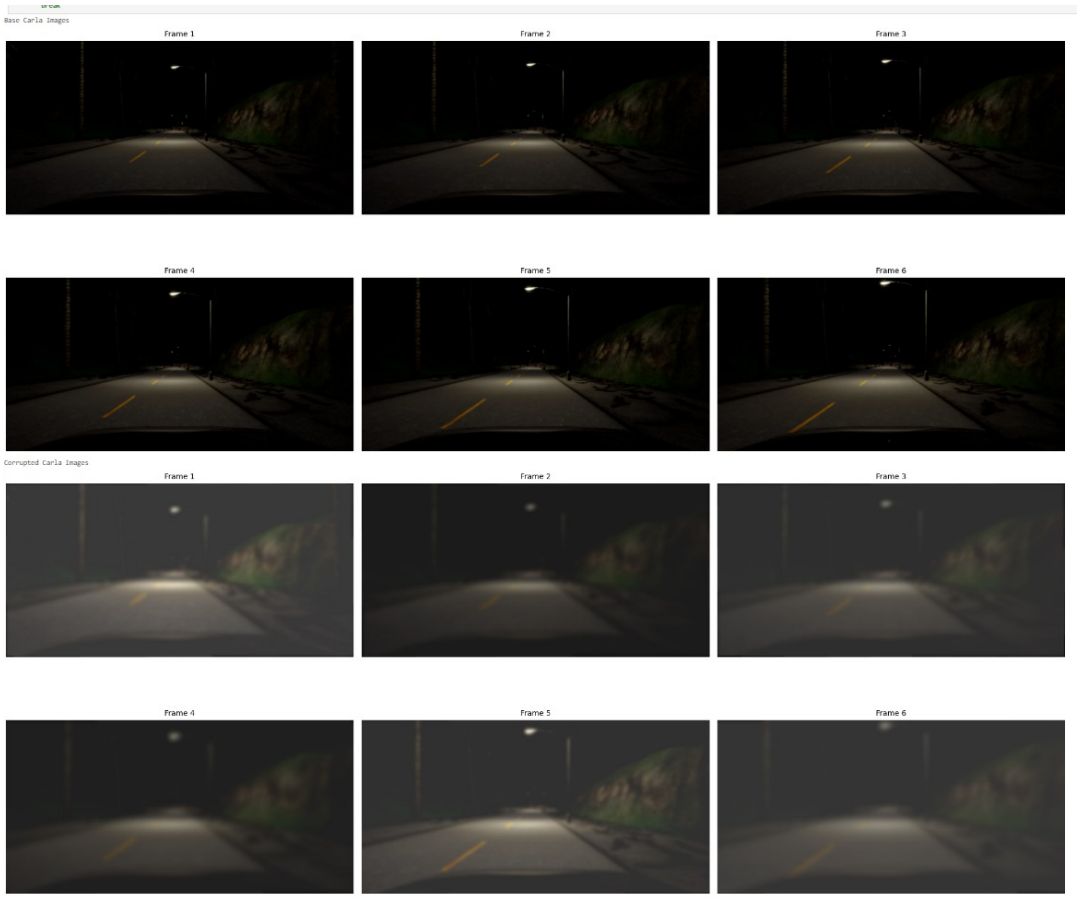


Fig. 1: Base CARLA on Top 2 and Corrupted CARLA on Bottom 2.

The second one is the *Corrupted CARLA dataset* where the 6 frames are corrupted by additive Gaussian noise, changing illuminations of images, adding blur effect, and doing illumination changes. The SequentialImagesDataset class defined in our code handles the complex task of loading and pre-processing data for models. It loads input Images and optical flow files, loads the camera extrinsic for the sequence, segmented images, and depth images, and converts them to tensors to be used for further processing.

When initializing the dataset for training, validation, and testing we are also doing corresponding transforms in the images. It ensures we are resizing images to 256x512, converting RGBA images into RGB and converting into PyTorch tensor, scaling its values to the range [0, 1]

Encoder- In the encoder, first we down sampled images and created a feature map to further pass into the next components of the architecture(2). we use sequential layers of Conv2d followed by Relu Activation function and MaxPooling2d layer, which reduces the spatial dimensions of the input while increasing the depth of feature maps. There is only one input going in encoder which is the input images tensor of shape [1, 3, 256, 512] where 1 is the batch size 3

is number of channels and 256, 512 is the height and width of images in tensor. This encoder will generate a feature map tensor with 64 channels.

3.1 Geometric Filter

ConvRNN In the ConvRNN module we extends the conventional LSTM framework by integrating convolutional structures within its design, allowing it to capture spatial hierarchies and temporal dynamics effectively. There are two input going in ConvRNN, first is camera encoder's output which we flattened and the final input going in ConvRNN had shape [1, 6, 64, 64, 128]. The second input was previous feature state which had shape [1, 64, 64, 128]. Such ConvLSTM architecture is suited for task like video frame processing (3). Components of ConvLSTM:

- **Initialization** : It accepts the number of input channels (inchannels) = 64, the number of hidden channels (hiddenchannels) = 64, the kernel size (kernelsize) = 3, and the stride = 1.
- **Convolution Layer (self.conv):** We are using here one convolutional layer. In the convolution process, involves sliding a filter or kernel over the input image to produce a feature map. If we look at the code the 2D convolutional layer is concatenating the input and hidden state to four times the hidden channels. This expansion is to create four different gates (input, forget, cell, and output gates) that are integral to the LSTM's functioning.
- **Forward Pass**:Here we processes sequential input data by iterating over each time frame. At each time step, here we combines the current input with the previous hidden state and applies convolutional operations to generate the LSTM gates.So with the help of these gates we update the cell state and the hidden state.We pass inputs as 5-dimensional tensor X with shape (batchsize, seqlen, channels, height, width), which are sequence data of Images. At the output side we have,"Output" which is a collection of hidden states for each time step.It basically represent temporal changes of images.We also have the (h-t, c-t) output representing the final hidden and cell states after processing the last element in the sequence.
- **Hidden State Initialization**: We have initializes the hidden and cell states to zero at the start of processing a sequence.

Depth Maps: The DepthDecoder is a convolutional neural network (CNN) architecture for upsampling encoded feature maps to generate depth maps with a higher spatial resolution, to match the spatial dimensions of ground truth depth data (4). We have done 2 upsampling steps and used ReLu activation function here. And also at the end added a convolutional layer (Conv2d) which adjusts the output from the last upsampling block to ensure the output depth map has the correct number of depth channels as specified by depth channels. So essentially our Depth Decoder effectively learns to predict depth at a pixel-wise level, helping to understand the scene geometry. The input tensor shape going in depth decoder was [6, 64, 64, 128](5).

Decoder: The Decoder module here is now doing a reconstruction of high-resolution data from lower-resolution encoded representations. This decoder was generating segmented images of input images. Here we have included two sets of convolution, batch normalization, and ReLU activation layers. In upsampling, for both layers, we used Bilinear upsampling if true, else Transposed convolution is false (6). The choice between bilinear upsampling and transposed convolutions is to have a balance between computational efficiency and the fidelity of the upsampling process. To summarize why we are doing the decoder part is to reconstruct detailed, high-resolution segmented images from compressed, lower-resolution feature maps.

3.2 Ego-Motion Filter

The Ego-Motion Filter is part of the neural network architecture to understanding and encoding the motion of the camera (or ego-vehicle) within the environment.

Motion Encoder: The Motion Encoder is to augment feature maps with motion information derived from optical flow. By merging motion cues with existing feature representations it is able to interpret dynamic scenes. We used 2 inputs here one is the current set of features that we got from the first encoder having tensor shape [1, 64, 64, 128] (5) and the second is a separate input of features at t-1, for this I used dataset's optical flow files to get this tensor, it had shape [1, 2, 512, 1024](5), which provides motion information between consecutive frames, capturing object and scene movements.

To ensure compatibility between the 2 inputs, the optical flow is resized to match the height (H) and width (W) of the current features. We are using here bi-linear interpolation for that.Then the resized optical flow map and the current feature map are concatenated along the channel dimension. It is then passed to the convolutional layer, which produces a feature map that embeds both the original spatial characteristics and the encoded motion information. It is then used as an input for other Camera Encoder.

Camera Encoder: We will be now passing the output from motion encoder into Camera Encoder which had tensor shape as [6, 64, 64, 128](5). The basic function here is to distill raw or pre-processed visual data into a more compact and informative representation that encapsulates camera-related attributes. Here we are using a 3 layer convolutional layer and employ max pooling and adaptive average pooling to down sample the feature map and achieve a 4x4-size output. This output helps in understanding the camera's interaction with its environment.

RNN: Here we needed to model temporal sequences and predict the position and orientation of a moving camera. It uses RNN(LSTM) (3) to handle sequential data with long-range dependencies. Its architecture is that it has an LSTM layer at its core to process the processes the sequential input data.After the LSTM processes the input sequence, its outputs for each time step are passed through a fully connected (linear) layer.So if we look at the functionality the RNN takes a sequence of combined inputs like motion features + relevant data and processes it through the LSTM to capture temporal dynamics.

If we look at the code LSTM maintains a hidden state and cell state (hn and cn) which encapsulate the temporal information from the input sequence for maintaining a memory of past inputs and later for future predictions. The outputs here is a sequence of ego-motion predictions,for each time step in the input sequence.

Camera Decoder: The Camera Decoder decodes 1D sequential data into a structured 4D output,for camera pose estimation.We used here 2 linear layers and a ReLU activation to process input data, outputting a tensor reshaped to represent spatial features.

4 Training and Validation

4.1 Training

We have done three model training experiments One is Supervised pre-training and End to end Fine-tuning, the second is End to End training at the Corrupted CARLA dataset and the third is End to End training at the base CARLA dataset.

- Supervised pre-training and End-to-end Fine-tuning- In this we are pre-training the dataset with the encoders and decoders of the model architecture without the RNN filters. In the fine-tuning part, the pre-trained encoders along with the RNN filters are included to further finetune the training process of the model.
- End-to-End training at Corrupted CARLA - In this experiment complete architecture(including RNN filters) is trained at the corrupted CARLA dataset. There is only one single training here. We did not fine-tune this experiment.
- End-to-end training of Base CARLA - Similar to corrupted Carla we have done the training with Base CARLA dataset using complete architecture components. The outputs of these results are discussed in detail in the results section.

Training implementation: The SegmentationModelTrainer class trains multiple interconnected models that are components of a larger system designed for image segmentation, depth estimation, and camera pose estimation. The specific models trained includes Encoder model that generates feature maps from input images,ConvLSTM model that processes the sequential data encoded in feature maps,Depth Decoder, Segmentation Decoder,Motion Encoder,Camera Encoder,Ego Motion RNN (Recurrent Neural Network) and Camera Decoder that predicts camera poses from the Ego Motion RNN outputs.

We are training the model for 5 Epochs.The batch size is 1, which contains around 8982 images.The optimizer we are using is Adam and the learning rate is 3e-4. Back propagation is used in our SegmentationModelTrainer's learning process, to optimize image segmentation, depth estimation, and camera pose estimation by iteratively minimizing loss through gradient descent.

We have calculated the Segmentation loss, depth loss and Camera loss converted into tensors and passed outside to visualize the individual losses later.We tried to pass values to Tensor Board here(as can be seen in the code), but later withdrawn due to some errors.Also at the end of training we have saved all the models in a Models/BaseCarla folder.

We have created a fit function fit() in our training pipeline, to provide a structured approach to model training and validation. By iteratively training the model, evaluating its performance, and then adjusting based on feedback from both training and validation datasets. It has given a better structure to our coding process.

Loss Equations are:- *Cross Entropy Loss* is for the segmentation task:-

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(p_{i,c}) \tag{1}$$

where N is the number of samples, C is the number of classes, $y_{i,c}$ is a binary indicator if class c is the correct classification for sample i, and $p_{i,c}$ is the predicted probability that sample i is of class c.

In our code we used the nn.CrossEntropyLoss class in PyTorch

Mean Squared Error (MSE) for camera pose estimation:

$$L_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2}$$

The *depth estimation loss for Depth Maps*, is defined as:

$$L_{\text{depth}} = \frac{1}{N}\sum_{i=1}^{N}|\log(\max(y_i, \min\_value)) - \log(\max(\hat{y}_i, \min\_value))| \tag{3}$$

L(depth) is depth estimation loss,N is no of pixels in image,y(i)is true depth value,y(hat) is predicted value,min value is constant to ensure the argument of the logarithm is never zero or negative. In our implementation we used torch.clamp in pytorch.

4.2 Validation

We are doing Validation for a single epoch, to assess the model's performance on new and unseen data. This is needed for evaluating the generalizability of the our trained model.When we are processing the dataset we are not updating weights here, that's the key difference with training.

In the validate epoch method,we are initializing various list to capture the performance of data.Next in the batch wise processing forward pass computation similar to training is done. But we are calculating the accuracy and loss measurement without back propagation(only in training), to show where model excels or may require further refinement.

Mean accuracy (mAcc) and Mean Intersection over Union (mIoU) are calculated here along with other computed losses.Also the aggregation of mean losses and performance metrics at the end of the validation process offers a holistic view of our model's efficiency.

Also note we are using a ProcessSegmentationTargets() to ensure that the segmentation targets(ground truth data) are in the correct format and dimension to ensure ensures smooth operation of loss computations and model training.Example for targets that are already in a 4D format (batch size, height, width, channels), it removes the channel dimension if it's not necessary for the loss computation.

5 Results

5.1 Supervised Pretraining, End-to-End Fine-Tuning

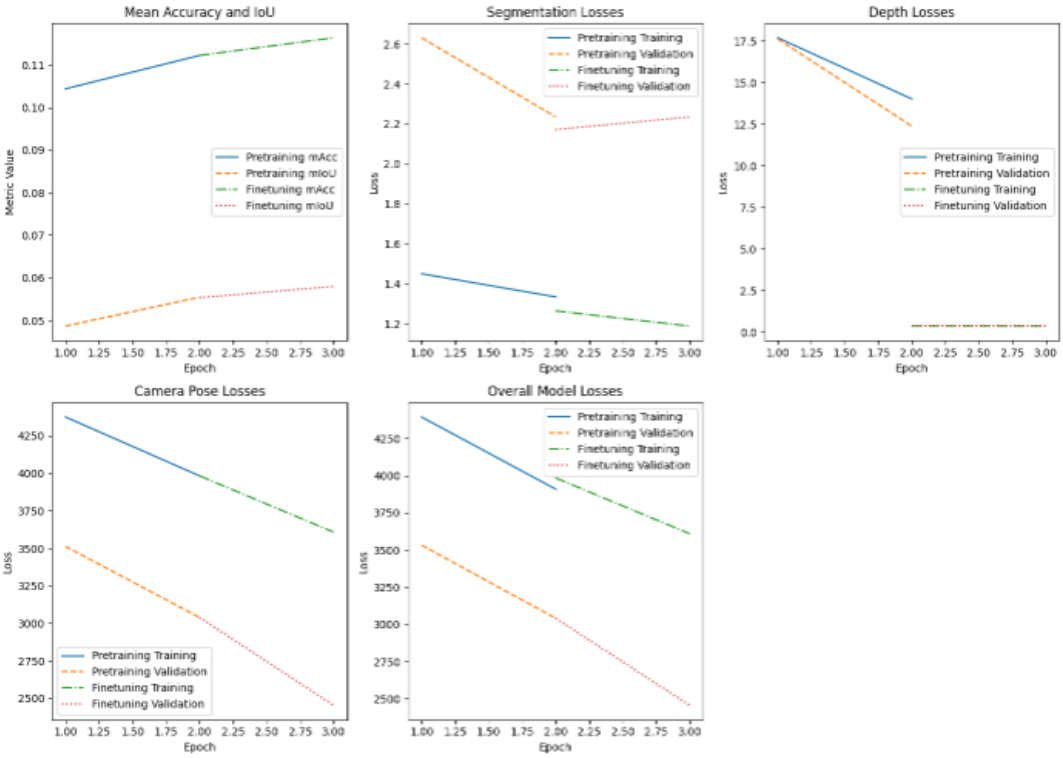


Fig. 2: Segmentation, Depth, Camera losses for Supervised Pretraining and Fine tuning.

In Fig 2, Pre trained and Fine tuned model results are shown. We have done 2 epochs for both.In the Segmentation and Overall Model loss we see a discontinuity or sudden break in loss values, because of fine tuning model starts from the saved pretrained models.Increasing mean accuracy is also a good sign which shows model is learning.For the depth loss, loss is decreasing during pre training,but in fine tuning we see a constant value, no fall in loss. In the camera loss part we see a good continuity in loss values.

Metric	Epoch 0	Epoch 1
Train Segmentation Loss	1.4492	1.3333
Train Depth Loss	17.7049	14.0292
Train Camera Pose Loss	4374.9746	3982.8711
Train Total Loss	4394.1289	3909.3235
Val Segmentation Loss	2.6315	2.2348
Val Depth Loss	17.6350	12.3947
Val Camera Pose Loss	3512.8877	3038.9045
Val Total Loss	3533.1543	3041.1557
mAcc	0.1044	0.1122
mIoU	0.0486	0.0554

Table 1: Pre tained Model Summary of losses and accuracy metrics over epochs.

Metric	Epoch 0	Epoch 1
Train Segmentation Loss	1.2633	1.1869
Train Depth Loss	0.3741	0.3686
Train Camera Pose Loss	3982.8711	3607.7681
Total Loss	3984.5090	3609.3235
Val Segmentation Loss	2.1717	2.2348
Val Depth Loss	0.3924	0.3947
Val Camera Pose Loss	3038.9045	2452.8528
Val Total Loss	3041.4690	2455.4822
mAcc	0.1122	0.1164
mIoU	0.0553	0.0580

Table 2: Fine-tuned Summary losses and accuracy metrics over epochs.

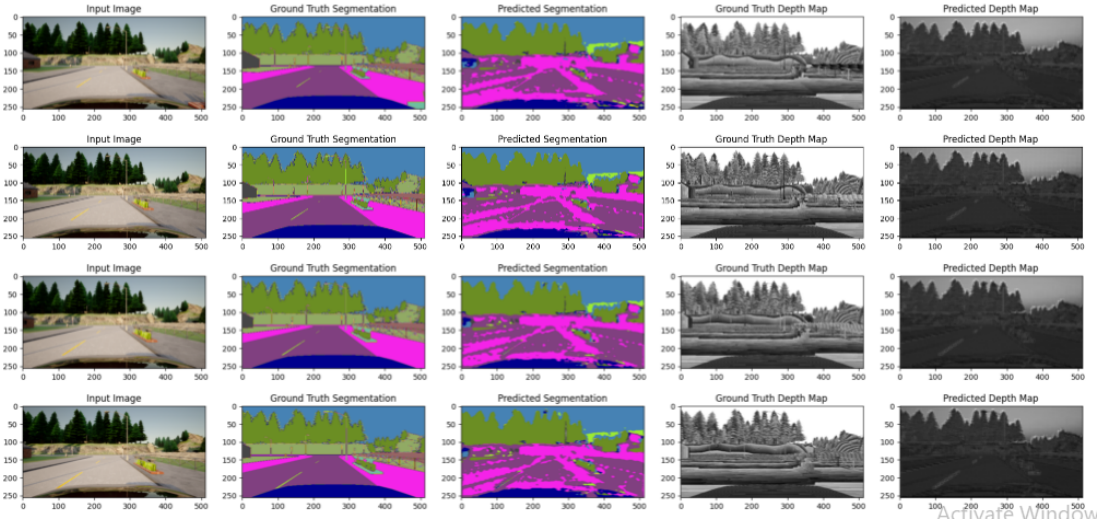


Fig. 3: Image comparison after Pre-training

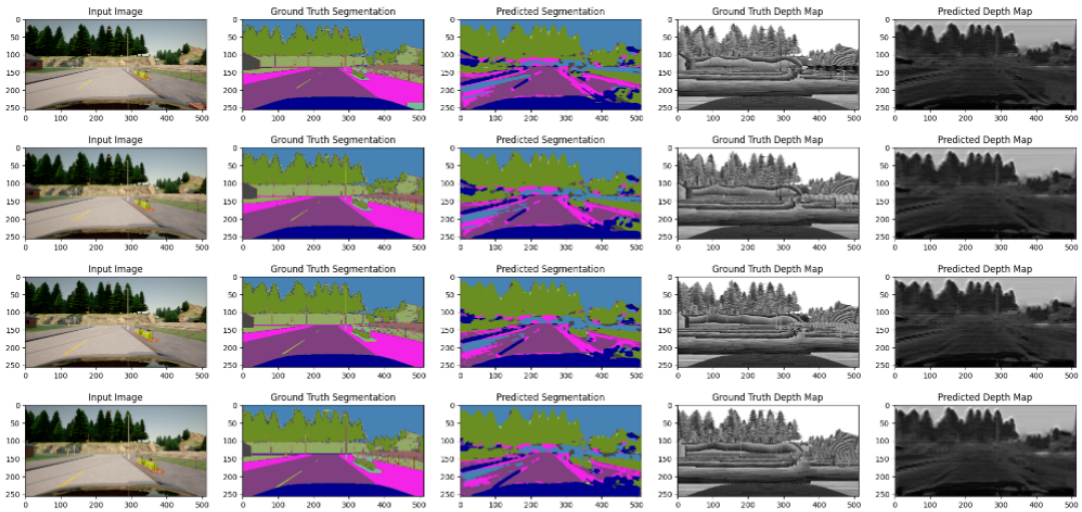
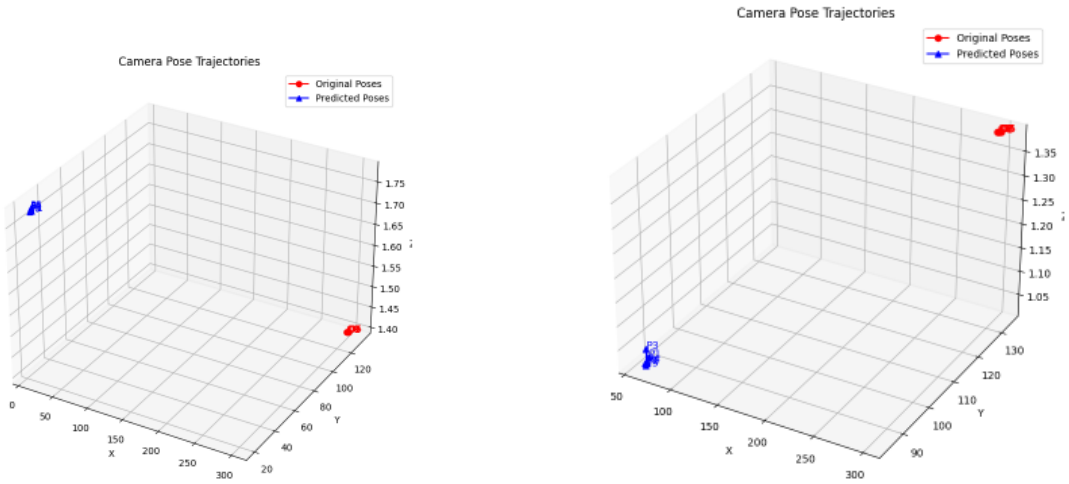


Fig. 4: Image comparison after Fine tuning

In fig 3 and fig 4 , the image comparison after Pre training and fine tuning is show.The input image is show,the ground truth segmentation and generated segmentation images and also the ground truth depth image and generated depth images in grey scale.This is shown for 4 images in a sequence.These outputs were better compared to our training with Base CARLA dataset.



(a) Pre trained Model

(b) Fine tuned Model

Fig. 5: Camera Pose

In fig 5 we noticed that in Pre trained model predicted trajectory appears to deviate significantly from the original path,showing lesser accuracy.But in fine-tuned model it is closer to the original trajectory.It is closer to the ground truth.The interpretation is that due to additional training fine-tuning phase has enhanced the model's understanding of the scene dynamics and the motion of the camera better.

5.2 End to End Training at Corrupted CARLA

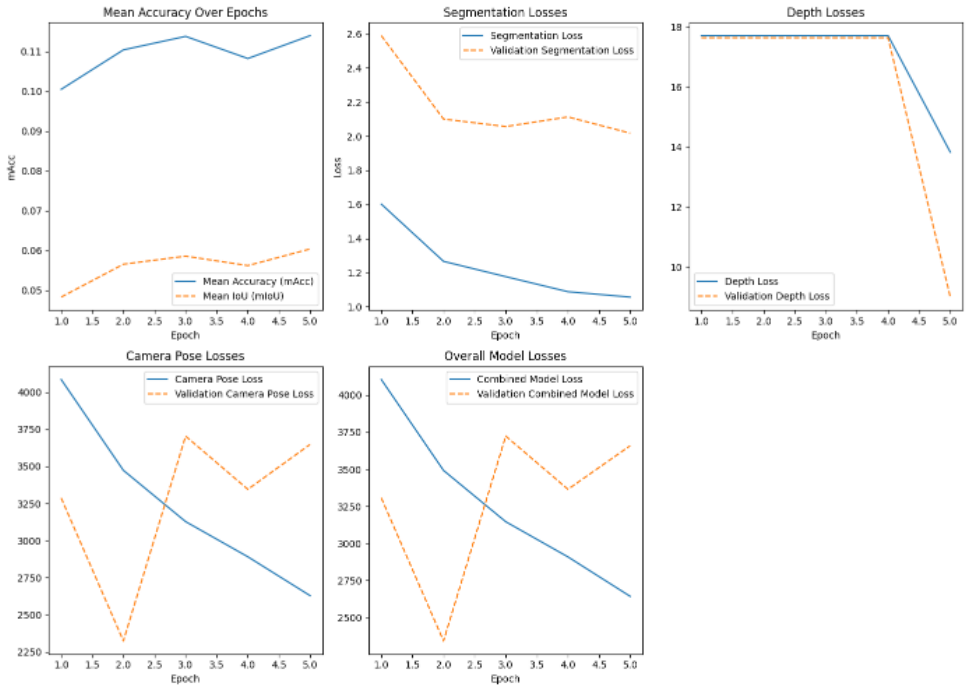


Fig. 6: Segmentation, Depth, Camera losses-Corrupted CARLA.

In fig 6 .Mean accuracy's are improving showing good performance.In segmentation and depth loss, the losses are decreasing.For depth loss,we see that after the 4th epoch a sudden fall in loss value.However in camera pose loss and overall model loss we see a sudden rise in loss after the 2nd epoch.This can be due to over fitting of training data learning its noise and peculiarities instead of generalizing the underlying patterns. Also compared to earlier tasks we are not pretraining the model here.We are here adding tables for End to End training and validation results for Corrupted CARLA dataset.

Table 3: Training Losses

Epoch	Seg Loss	Depth Loss	Cam Pose Loss
0	1.60	17.70	4085.47
1	1.27	17.70	3473.05
2	1.18	17.70	3127.50
3	1.09	17.70	2890.48
4	1.06	13.83	2628.48

Table 4: Validation Metrics

Epoch	Seg Loss	Depth Loss	Cam Pose Loss	mAcc	mIoU
0	2.59	17.64	3286.77	0.100	0.048
1	2.10	17.64	2323.04	0.110	0.057
2	2.06	17.64	3703.78	0.114	0.059
3	2.11	17.64	3345.04	0.108	0.056
4	2.02	9.00	3647.60	0.114	0.060

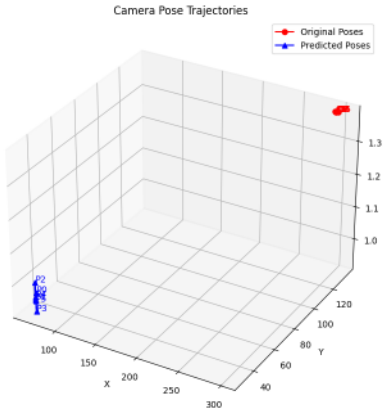


Fig. 7: Camera pose trajectory-Corrupted CARLA

In fig 7. similar to the earlier fine-tuned model, the Corrupted Carla model also shows the predicted trajectory is much closer to the original path showing closer to the ground truth.

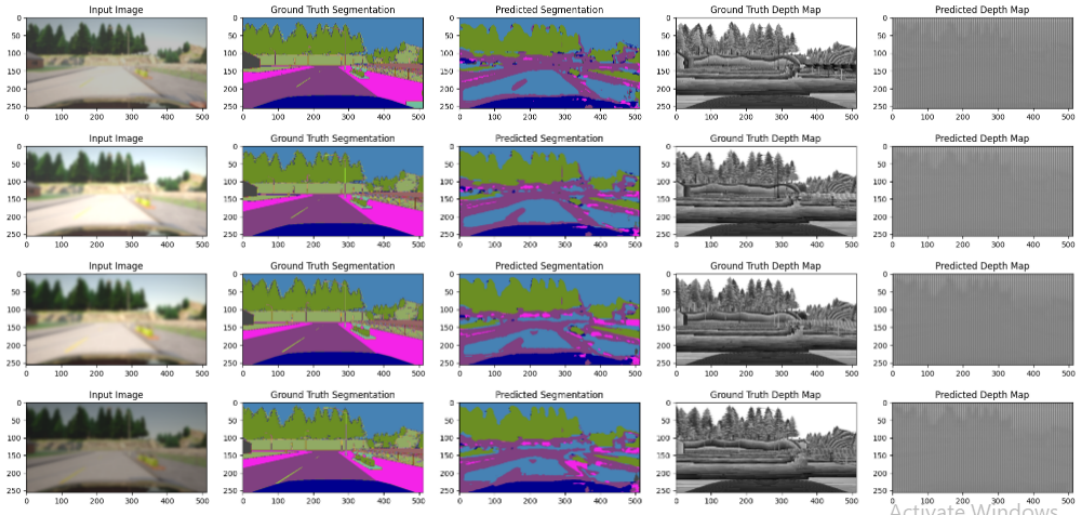


Fig. 8: Comparison of Orginal and Generated Image outputs-Corrupted CARLA

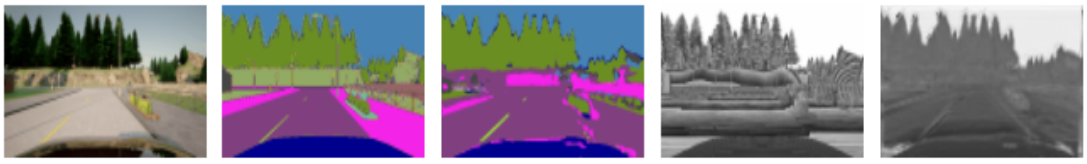


Fig. 9: Gif images of Corrupted CARLA

Fig 8 and Fig 9 shows the comparison image output and Gif image(single image here) generated for Corrupted CARLA.Gif sequence for image are show in our output datasets

5.3 End to End Training at Base CARLA

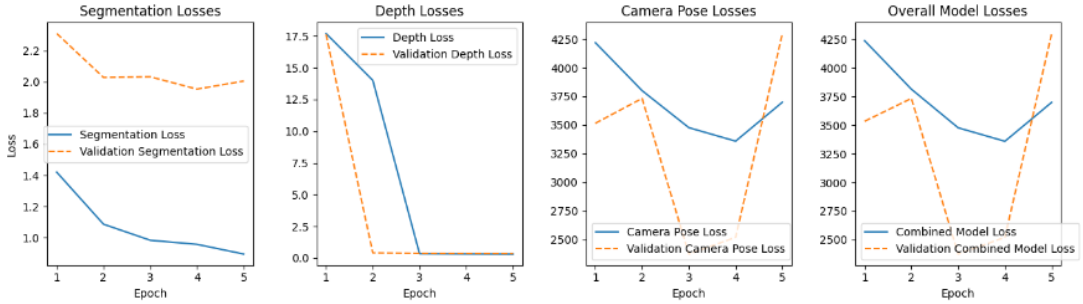
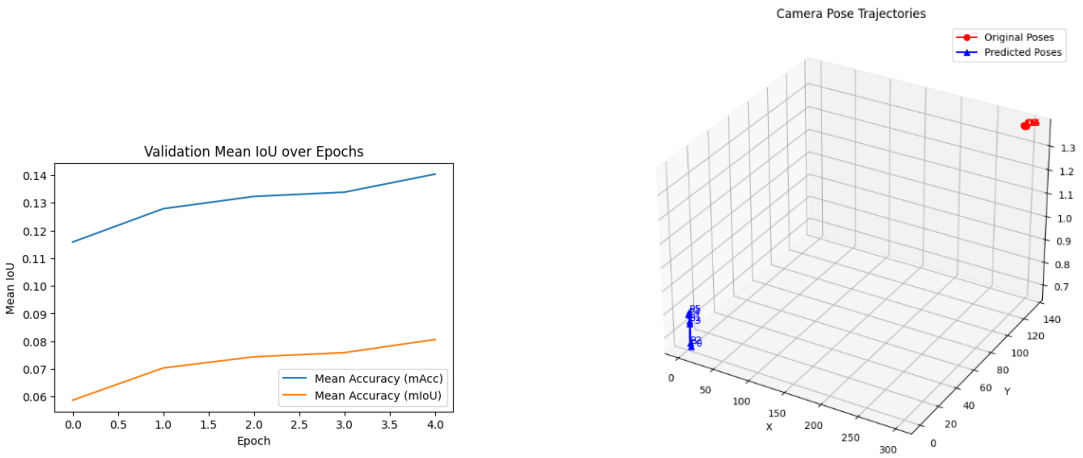


Fig. 10: Segmentation, Depth, Camera losses-Base CARLA.



(a) mAcc and mIoU values

(b) Camera pose trajectory

Fig. 11: Base CARLA-Mean Accuracy and Camera pose trajectory

Figure 10 shows the three losses and Fig 11 shows the accuracy's and camera pose trajectory for Base CARLA End to End Training.For segmentation loss more validation loss is there but for all others validations losses are lesser at least till the mid epochs.Fig 10 is more aligned with the training as show in Corrupted CARLA.

6 Conclusion

Experimenting our models in Supervised pretraining cum finetuning, End to end training with corrupted CARLA dataset and also base CARLA dataset has given exceptional results which varies based on the model structure and dataset.We could clearly observe how model performed well when pretrained with encoders and decoders in our fine tuning tasks.

Throughout this project, we faced multiple challenges such as data interpretation and loading, model complexity, and computational constraints. Yet, by iteratively refining our approaches and techniques, we managed to overcome these hurdles, leading to significant improvements in project completion and getting better results across all tasks.

This modeling philosophy not only elevates the accuracy of scene interpretation but also unveils new pathways for applying such integrated systems in areas such as autonomous navigation, augmented reality, and interactive robotics. This project realizing the vision of autonomous systems that can adeptly navigate and interact with their environment with unprecedented sophistication.

7 Contributions

Supervised pre training and End to End fine tuning implemented by Areesha Asif, End to end training with corrupted CARLA done by Abhishek S Pillai..Base CARLA implementation has been done by both. Training of models we done simultaneously by both team members to save time. Regular coordination and team meeting done for model architecture discussion ,code review and improving the model and results. Final project report done was done with close coordination by both team members each contributing their valuable information on all major sections. We would also take this opportunity to thank Prof. Dr. Sven Behnke and Angel Villar Corrales for mentoring and supporting us to complete the project on time.

Bibliography

[1] J. Wagner, V. Fischer, M. Herman, and S. Behnke, "Functionally modular and interpretable temporal filtering for robust segmentation," in *Proceedings of the British Machine Vision Conference (BMVC)*, Bosch Center for Artificial Intelligence and University of Bonn, 2018. https://www.ais.uni-bonn.de/papers/BMVC$_2$018$_W$*agner.pdf*.

[2] T. Ni, A. Norelyaqine, R. Azmi, and A. Saadane, "Architecture of deep convolutional encoder-decoder networks for building footprint semantic segmentation," *Scientific Programming*, vol. 2023, p. 8552624, 2023.

[3] C. Olah, "Understanding lstm networks." Blog post at colah's blog, Aug 2015.

[4] S. Paul, ""reparameterization" trick in variational autoencoders." Towards Data Science, Apr 2020.

[5] Charisoudis, "A simple conv2d dimensions calculator & logger." https://charisoudis.com/blog/a-simple-conv2d-dimensions-calculator-logger.

[6] S. Mehta, "Encoder-decoder networks for semantic segmentation," 2017.