BM 582 - HW 1

Emre Girgin - 2016400099

1. Introduction

Through this assignment, we are asked to train SC-SfMLearner with VR-Caps Unity dataset. This dataset is obtained in a virtual environment simulating endoscopy. Our goal is to conduct depth and pose estimation.

There are several configurations to be tested: the number of epochs, the learning rate, and the datasize. The number of epochs is either 10 or 20, the learning rate is either 0.002 or 0.0001, which is the default learning rate, and the data size is either 250 or 500. In total, there are 8 scenarios which are combinations of the options above.

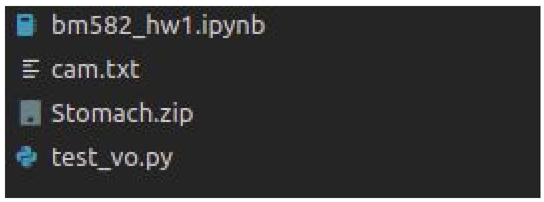
Note that, since the dataset is sequential, we can not shuffle the dataset, before we feed into the network. The train/val/test split can be independent but each of them has to consist of consecutive frames because each sample derived from 3 consecutive frames. (Actually 3 is an arbitrary number, it is written because it is the default value.)

Some important remarks:

- The encoding of the images in the dataset is PNG, but the repository wants the train and validation sets to be encoded in JPG and test to be encoded in PNG. The notebook transforms them automatically.
- The output paths in the "test_vo.py" file is hardcoded and causes some issues with our dataset, thus I'll provide an additional test_vo.py file in the submission. The notebook I

submitted with it automatically replaces the original file with the one I provided.

- To be able to run the notebook, the "Stomach.zip" has to exist in the path where the notebook is run. Due to limitations in the submission size, this file is not included. **Please add this file (Stomach.zip) where the notebook resides in.** Sample file structure:



- The cam.txt file is also provided. However, only the focal length and optical center parameters are used. Radial distortion coefficients are discarded.

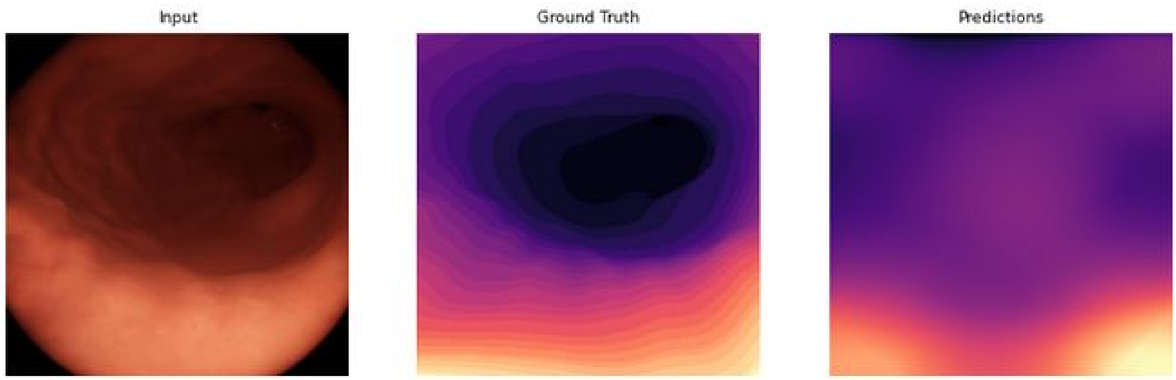
159.5906 (fx)	0	161.9297 (cx)
0	159.2241 (fy)	163.1686 (cy)
0	0	1

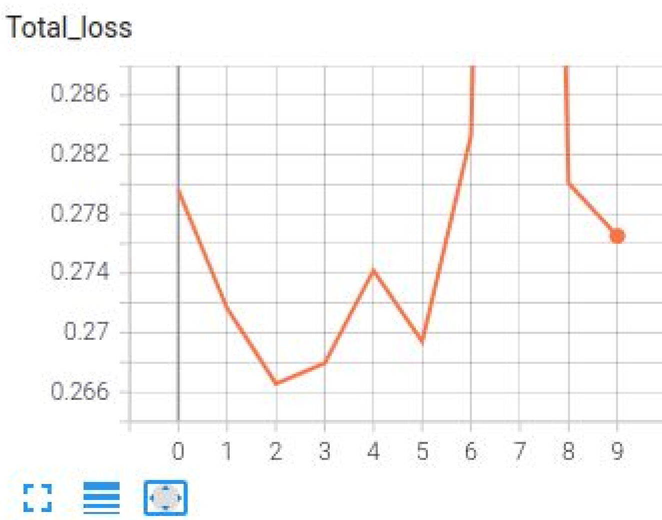
- The selection of the sub datasets which have 250 and 500 sizes, is done randomly but it is set using seed. (See create_dataset function in the notebook.)
- The notebook is prepared in Colab. It can be safely test there. (ofc, additional files are needed to be present as well)

2. Work Done

2.1. RMSE Scores

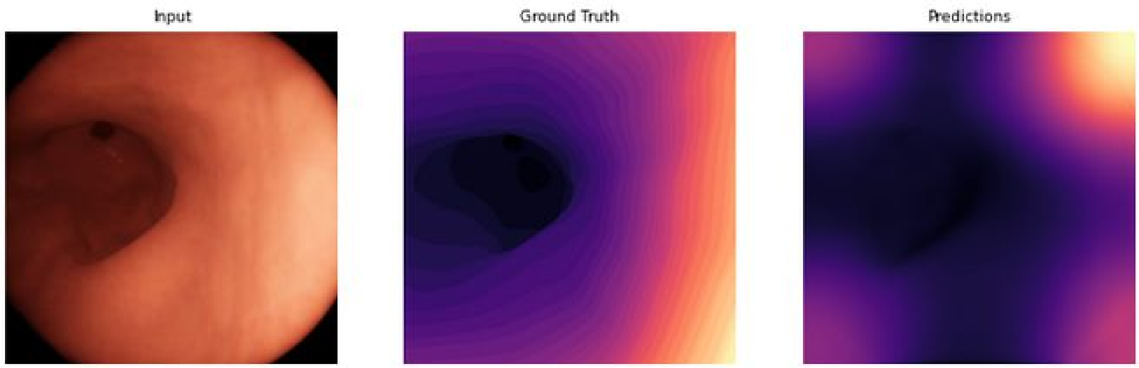
2.1.1. Configuration 1 (Epoch: 10, LR: Default, Datasize: 250)

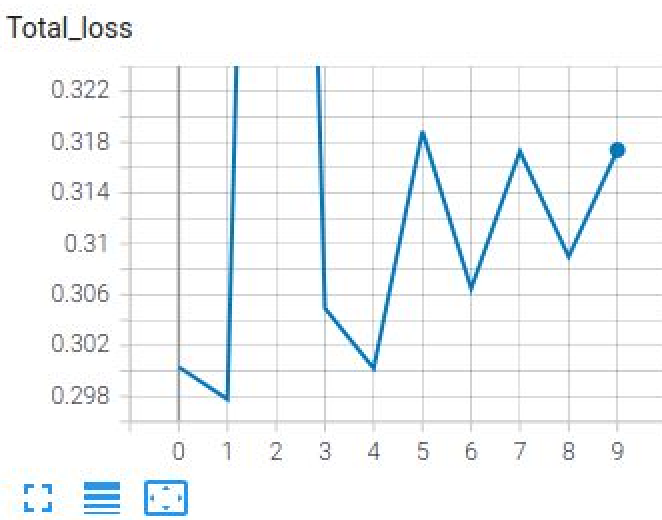




RMSE Score: 0.17163708737602407

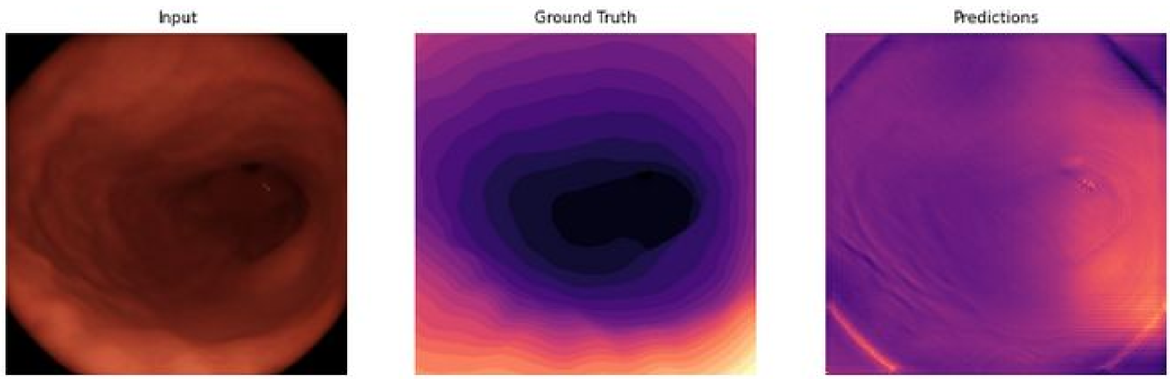
2.1.2. Configuration 2 (Epoch: 10, LR: Default, Datasize: 500)

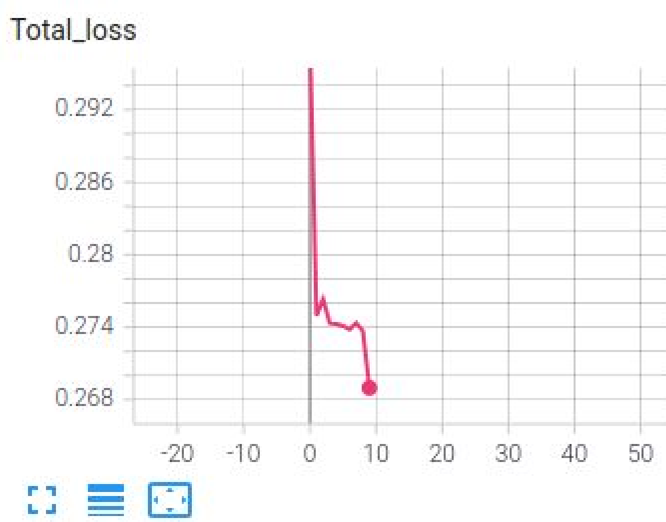




RMSE: 0.2318926980395691

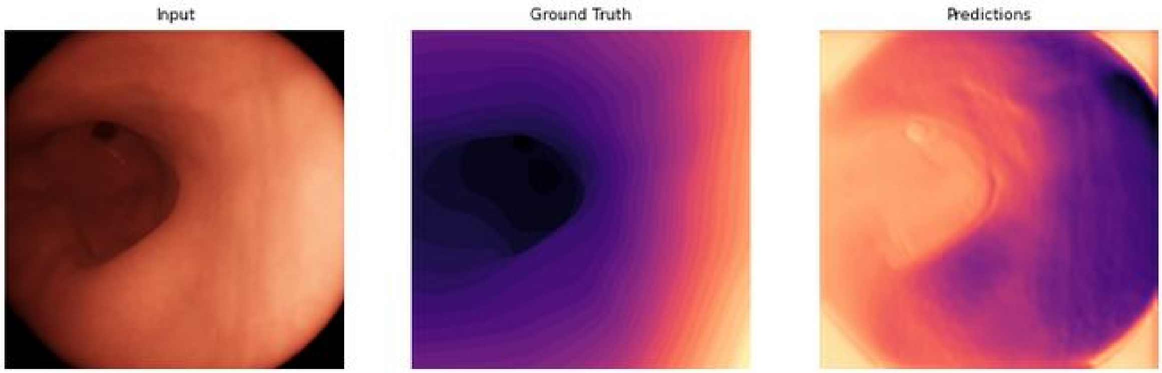
2.1.3. Configuration 3 (Epoch: 10, LR: 0.002, Datasize: 250)

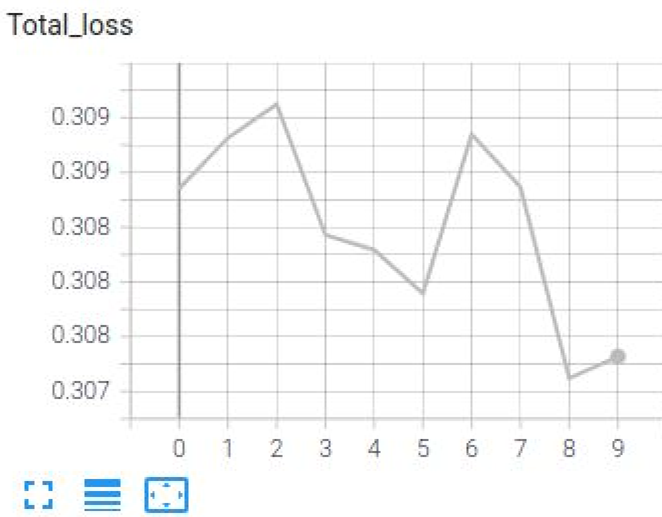




RMSE: 0.2484247726222485

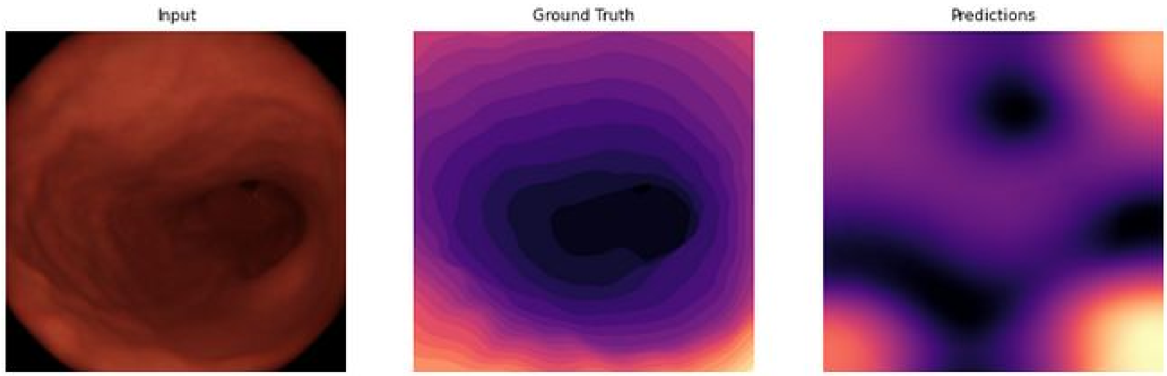
2.1.4. Configuration 4 (Epoch: 10, LR: 0.002, Datasize: 500)





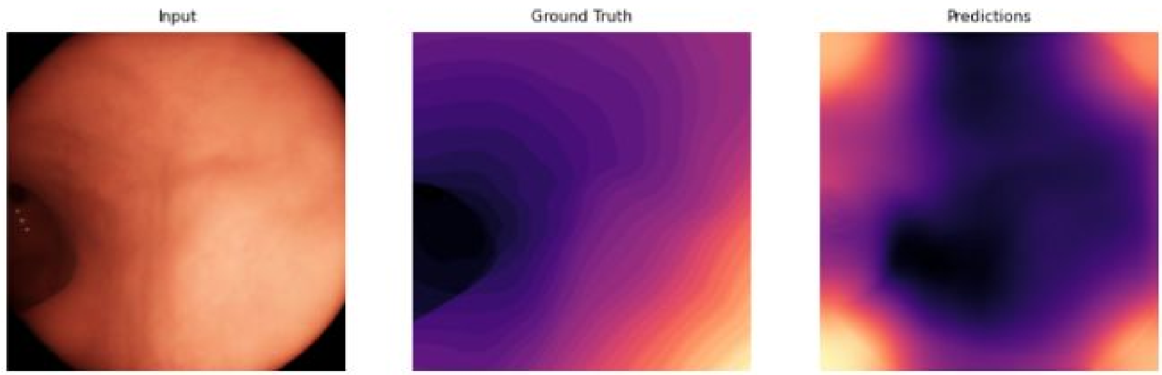
RMSE: 0.41068357416068646

2.1.5. Configuration 5 (Epoch: 20, LR: Default, Datasize: 250)



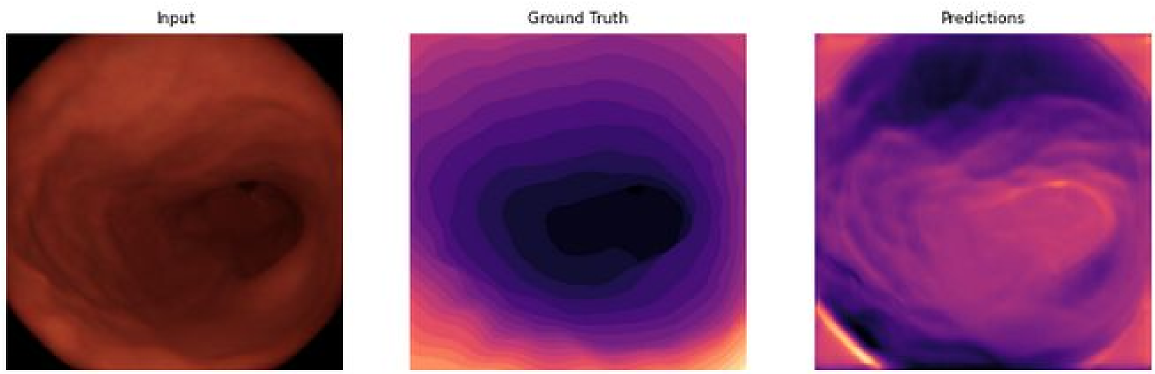
RMSE:0.21871928099819704

2.1.6. Configuration 6 (Epoch: 20, LR: Default, Datasize: 500)



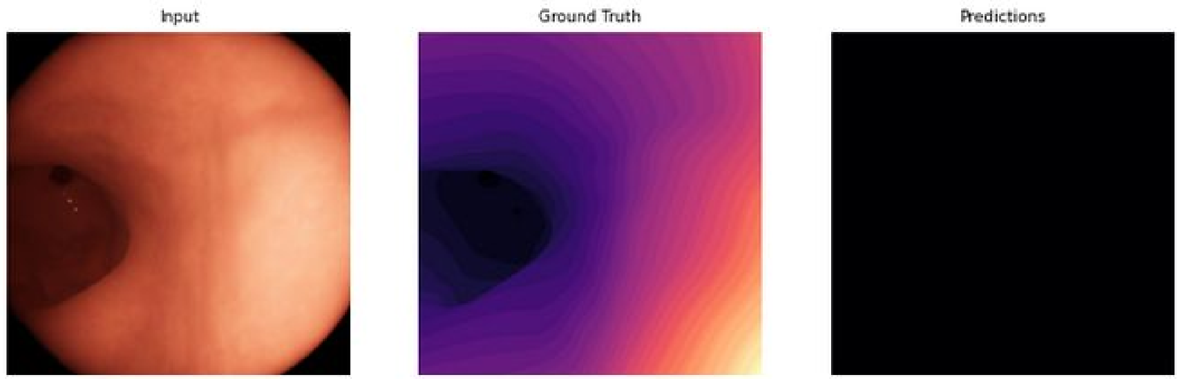
RMSE:0.26839096169001325

2.1.7. Configuration 7 (Epoch: 20, LR: 0.002, Datasize: 250)

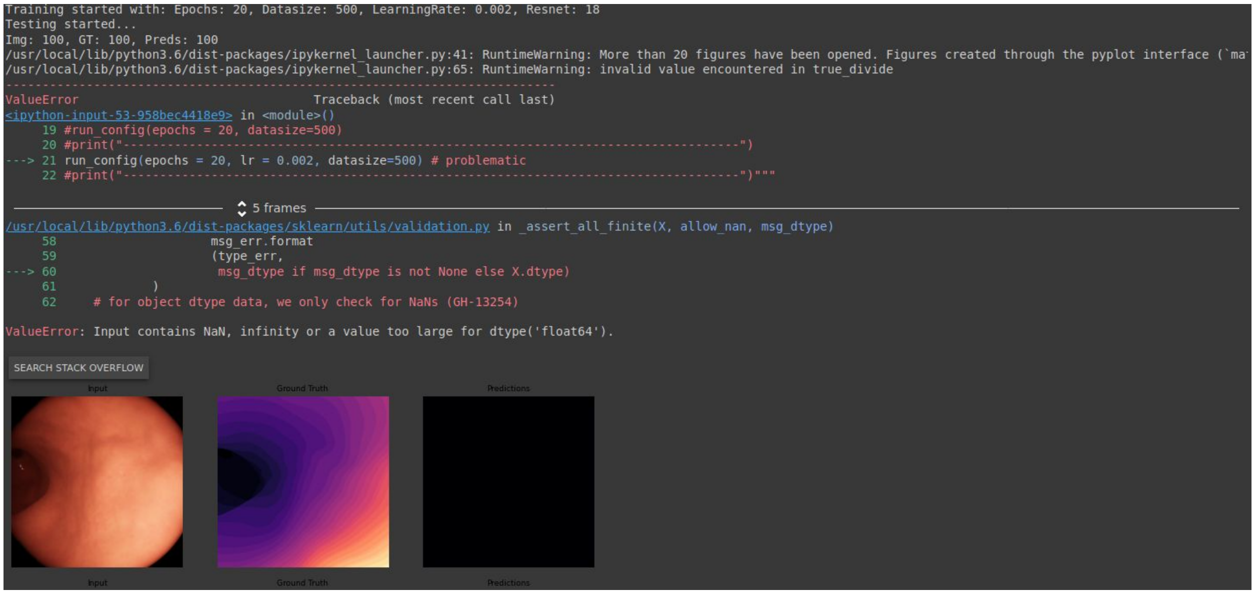


RMSE:0.38255379997192995

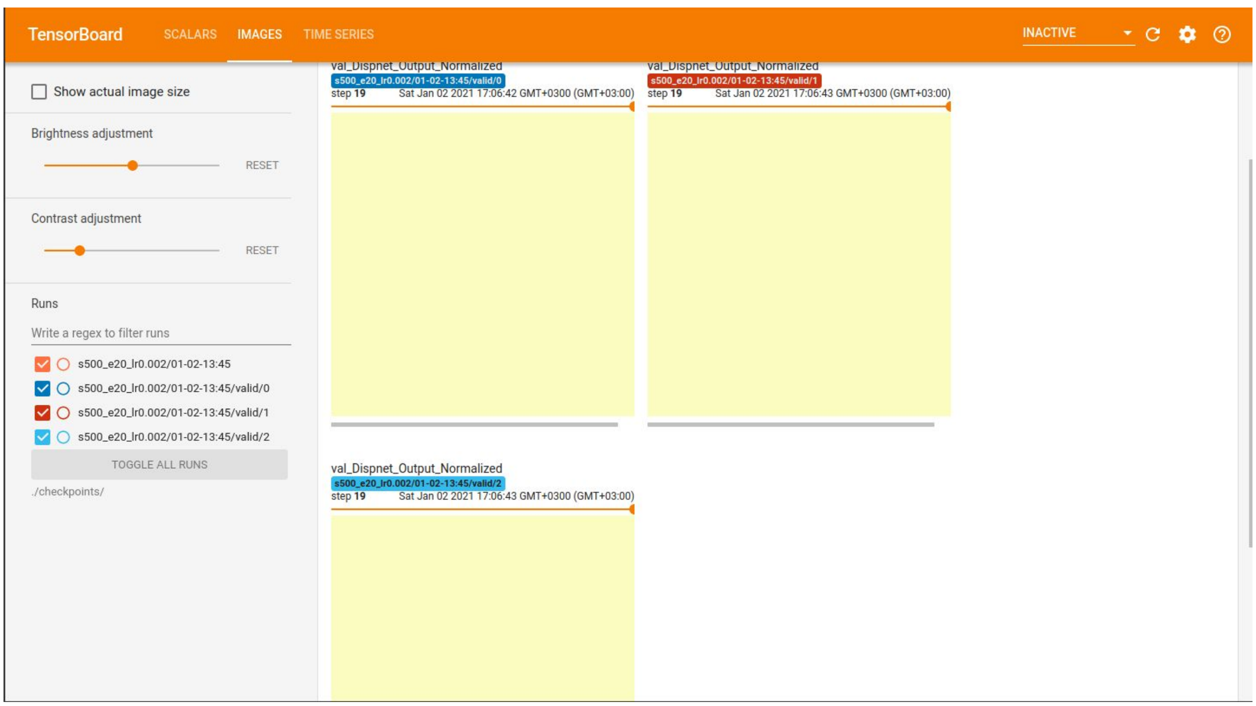
2.1.8. Configuration 8 (Epoch: 20, LR: 0.002, Datasize:500)



For this configuration, it seems learning rate is too big and the model diverges. Thus the predictions is too big that they do not fit into float64.

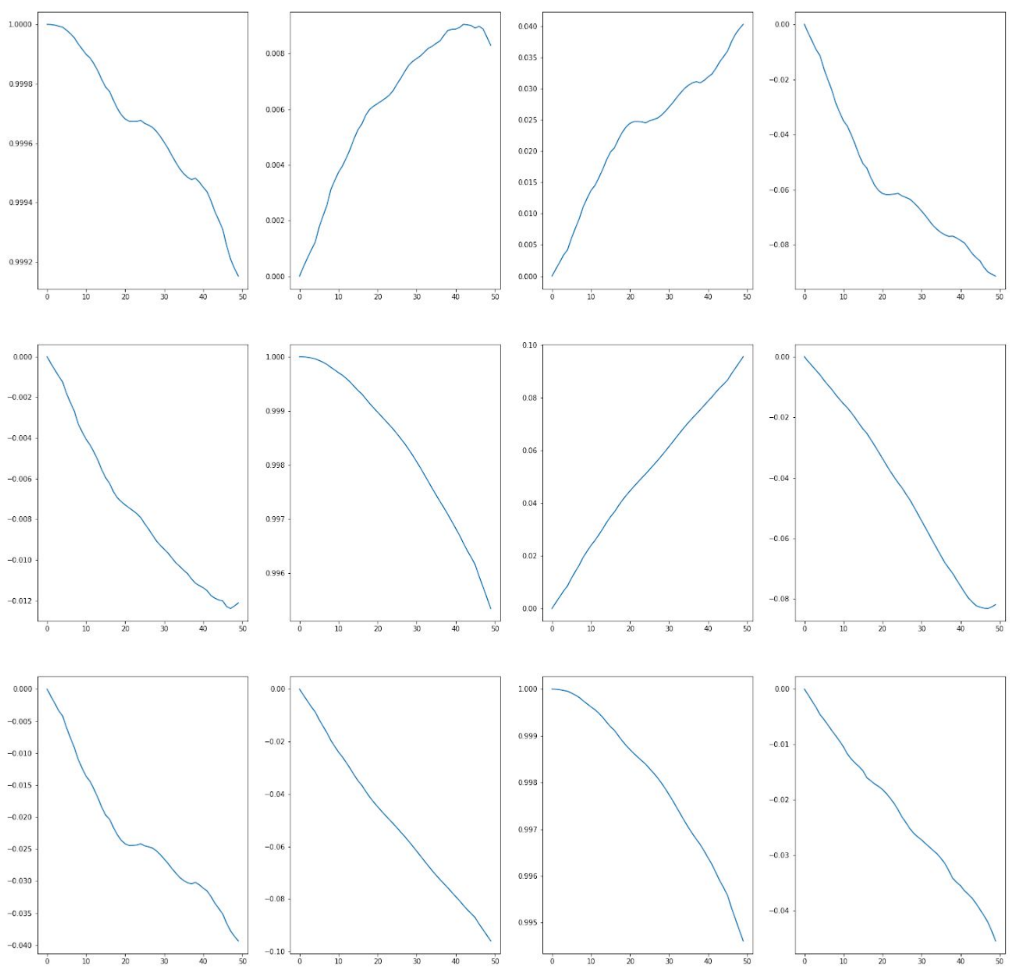


This is the some validation set outputs from Tensorboard:

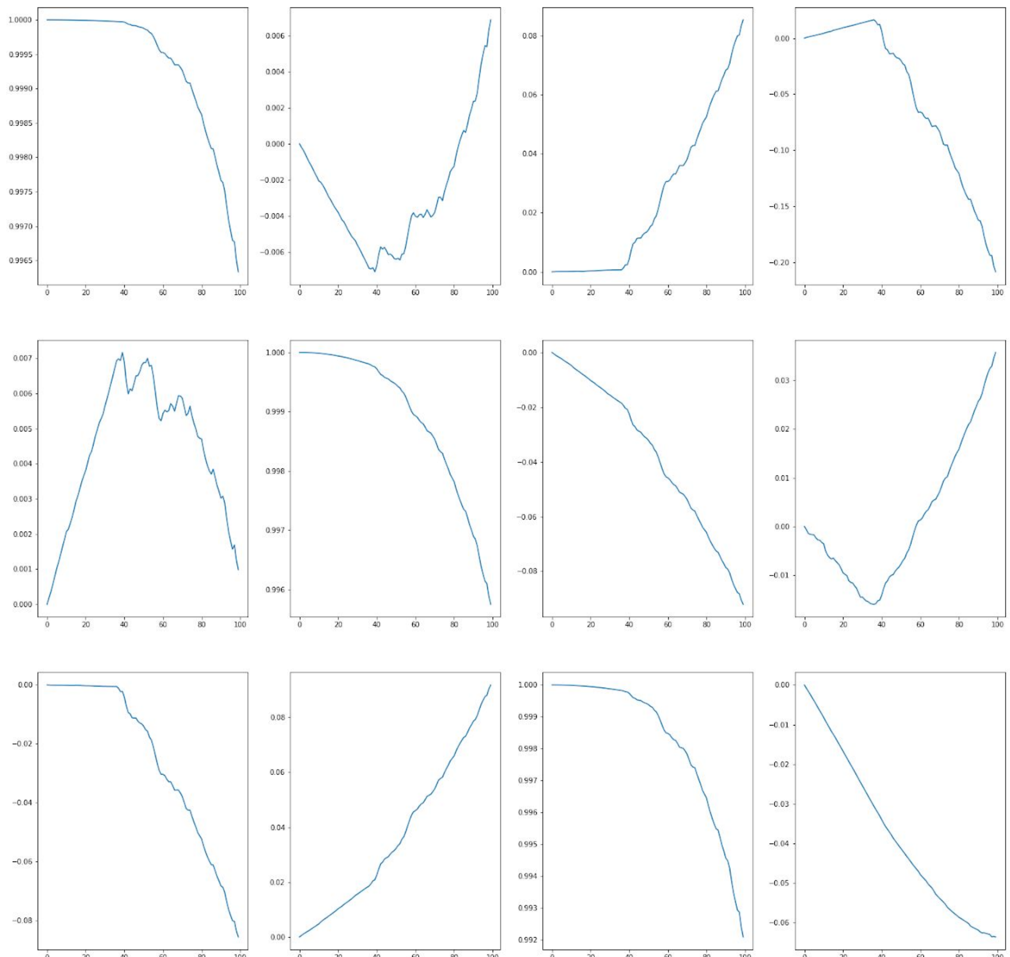


2.2. Trajectory Estimations

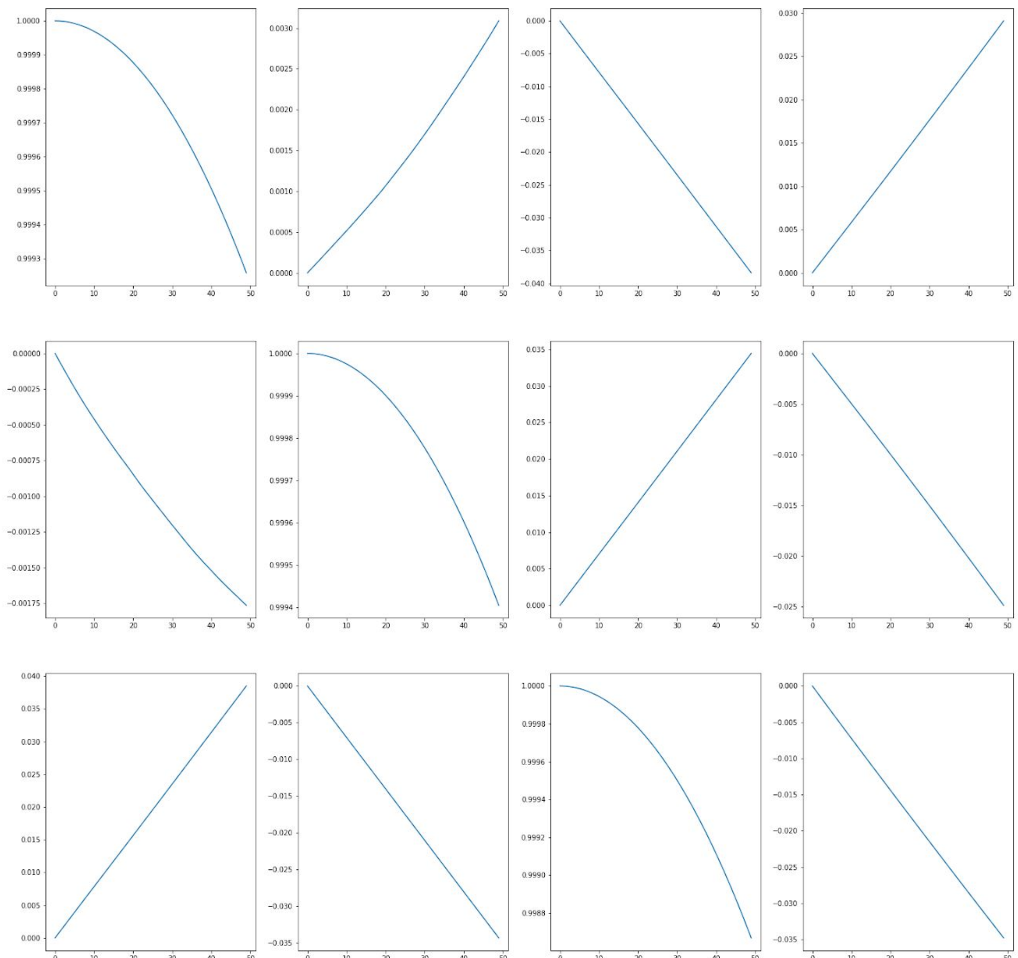
2.2.1. Configuration 1 (Epoch: 10, LR: Default, Datasize: 250)



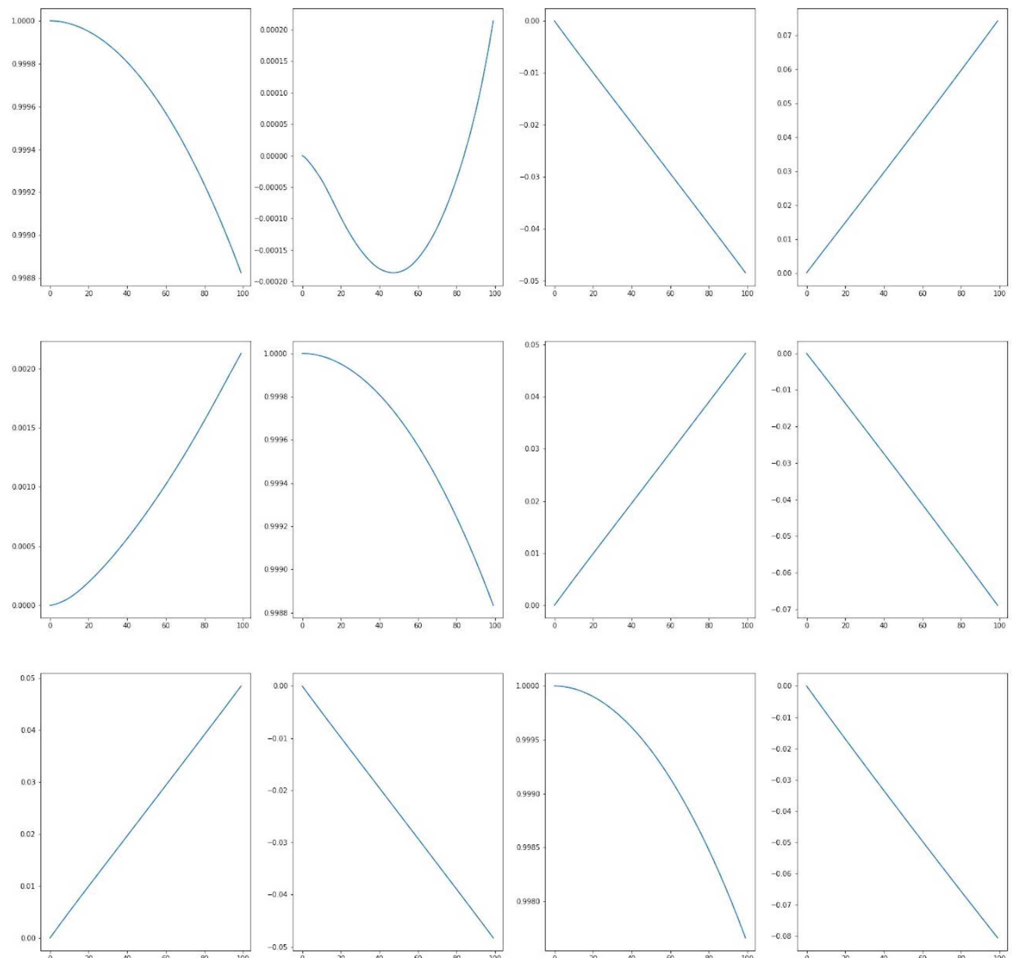
2.2.2. Configuration 2 (Epoch: 10, LR: Default, Datasize: 500)



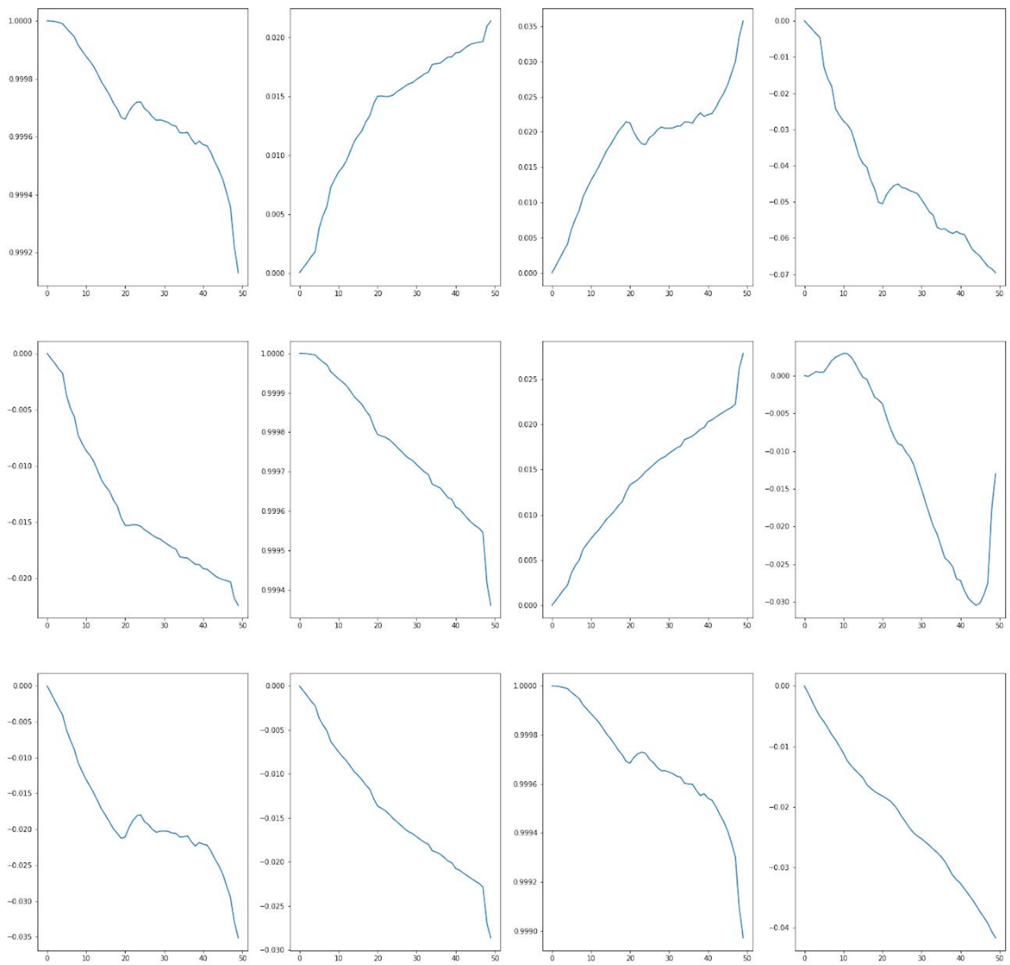
2.2.3. Configuration 3 (Epoch: 10, LR: 0.002, Datasize: 250)



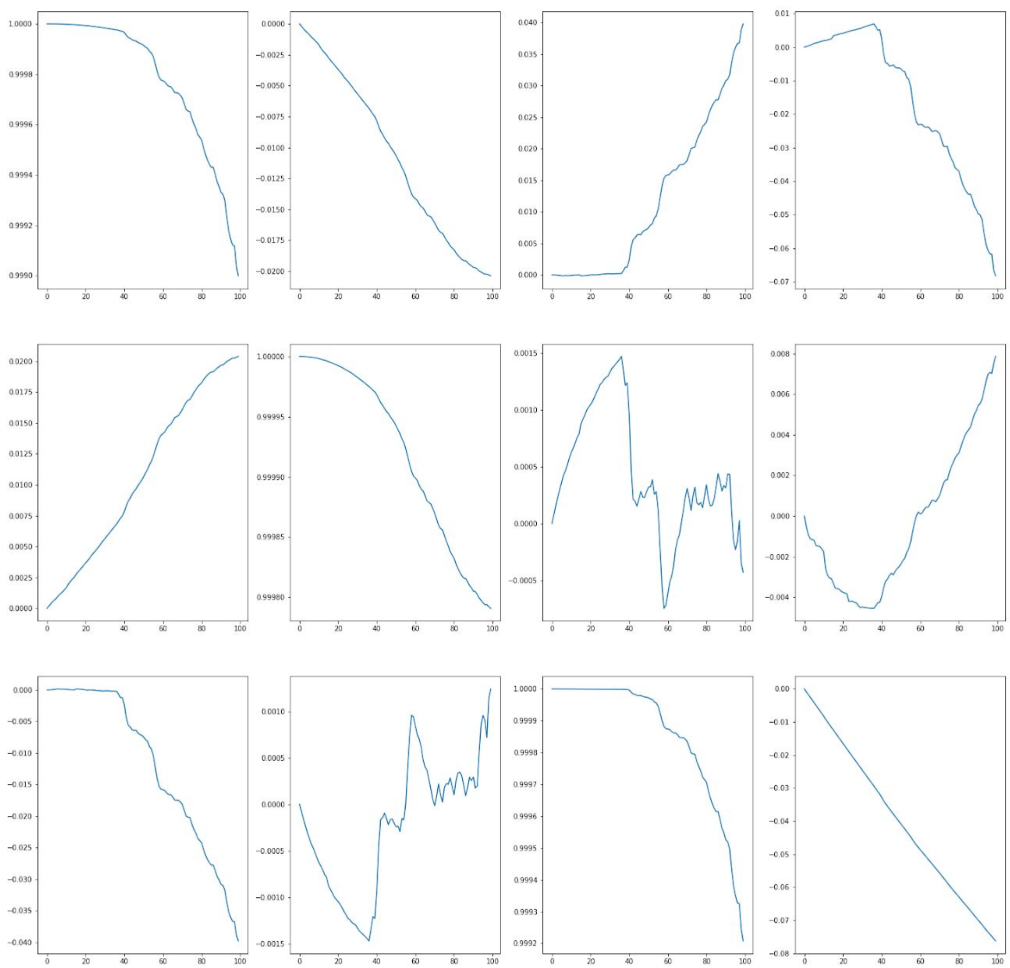
2.2.4. Configuration 4 (Epoch: 10, LR: 0.002, Datasize: 500)



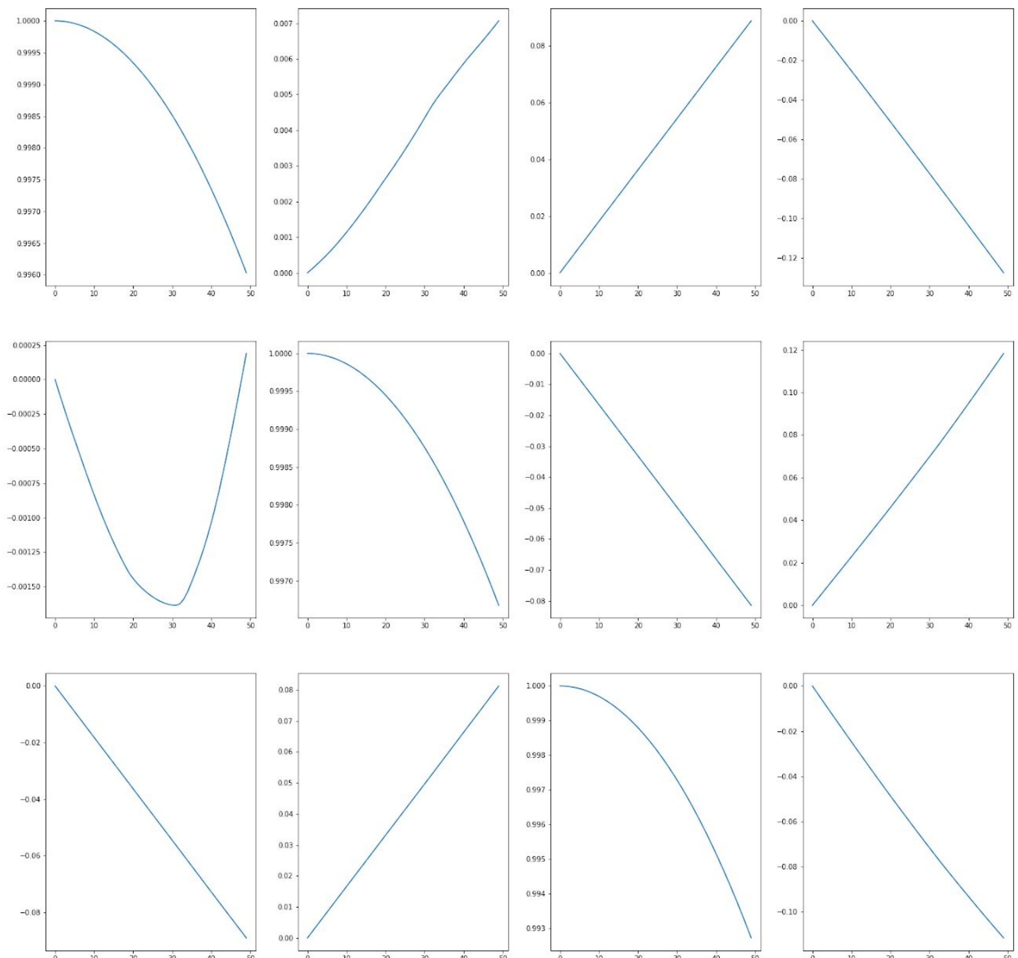
2.2.5. Configuration 5 (Epoch: 20, LR: Default, Datasize: 250)



2.2.6. Configuration 6 (Epoch: 20, LR: Default, Datasize: 500)



2.2.7. Configuration 7 (Epoch: 20, LR: 0.002, Datasize: 250)



2.2.8. Configuration 8 (Epoch: 20, LR: 0.002, Datasize:500)

Due to problems mentioned in 2.1.8 trajectories could not be plotted.

2.3. Loss Functions

There are four loss functions reported in the repository: Smooth Loss, Photometric Loss, Geometric Consistency Loss, and Total Loss.

2.3.1. Smooth Loss:

Smooth loss (a.k.a disparity smoothness loss) is used to calculate the difference between target image's depth and predicted depth. (See loss_functions.py Line132) It is calculated on the DispResNet.

2.3.2. Photometric Loss:

Photometric loss is the sum of the mean of the both reference and target images on a given mask. This mask is calculated via the inverse_warp2 function. (See inverse_warp.py Line230). The result of both networks, DispResNet and PoseResNet, are used to calculate this loss function. The SSIM, Mask, AutoMask, and Padding options are available. Padding option pads zeros while transferring the image taken from the camera to pixel plane. SSIM option calculates the similarity between target image and reference image and turns image difference score to a linear combination of the similarity score and itself. AutoMask option shirks some pixels of the mask obtained after the warping operation. The Mask option changes the mask accordingly the difference between projected depth and computed depth.

2.3.3. Geometric Consistency Loss:

This loss is calculated very similar to the Photometric Loss. The only difference is Geometric Consistency Loss is calculated on depths whereas, photometric loss is calculated on images.

2.3.4. Total Loss:

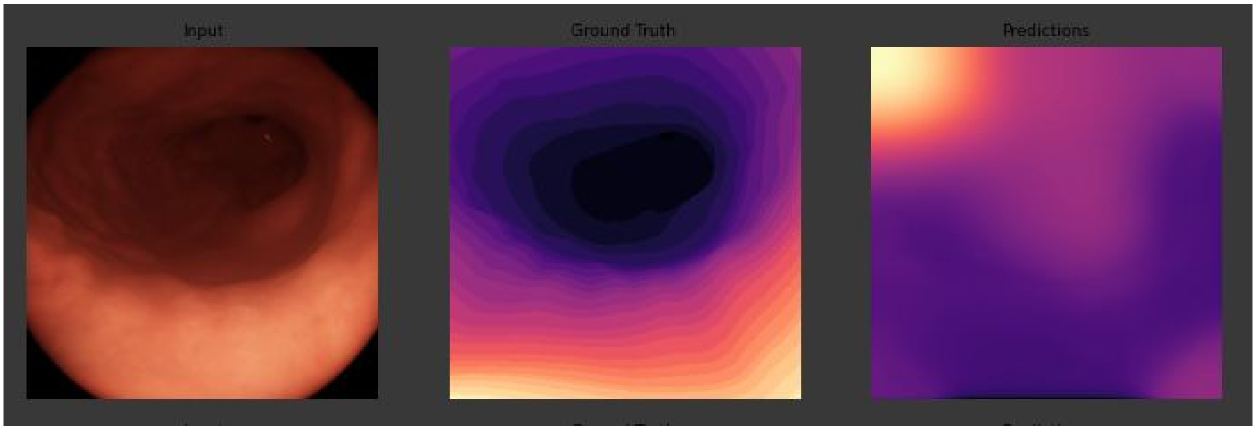
Total Loss is a linear combination of three losses above. Each loss has its own weight and these weights obtained from parameters of training. The default weights of them are 0.1, 1, 0.5, respectively. In our setup, we used those coefficients as well.

2.3.5. LPIPS Loss:

LPIPS stands for Learned Perceptual Image Patch Similarity. It is proposed in this paper to come up with a better similarity metric similar to human perception. In our outputs, it is not necessary to have exact matching with the ground truth. Since we are interested in the relative depth of the patches in the input, increasing the value of each pixel by 1 does not affect the result we are looking for. However, such shifts increase the result of some loss metrics such as L2. By using a metric assessing similarity in the level of human perception (like a doctor does), matters more at the end. This metric is already implemented here.

2.4. ResNet Architectures

2.4.1. Resnet 50



RMSE: 0.3001606674917958

3. Conclusion

Based on the predictions, the increasing the number of epochs and datasize looks like helping the predictions. However, increasing the learning rate seems affecting results in a worse manner. Probably the learning rate is too large so that weights do not converge to an optimal minimum, they rather diverge.

I think the best results obtained from configuration 6, which has 20 epochs training and datasize 500 with default learning rate. However, the best (the lowest) RMSE score is obtained in Configuration 1. This is why I think RMSE is not a good metric for evaluating this experiment.

Although I used the pretrained weights, 10 or 20 epochs are very small to have reasonable results. In addition, probably the pretrained weights are not trained for a dataset which comes from a similar distribution to VR-Caps, our dataset.

On the other hand, this assignment taught us to investigate, train, and evaluate a deep learning repository that we have never faced before. The results of different configurations are compared and some results are promising.

PS: Outputs are under,

- /SC-SfMLearner-Release/checkpoints

- /SC-SfMLearner-Release/results
- /SC-SfMLearner-Release/vo_results
- /SC-SfMLearner-Release/graphs
- /SC-SfMLearner-Release/rmse.json