# Deepslam Extensions

Adam Rivelli Ignacio Gavier Matthew Pearce

## Abstract

*Simultaneous Localization And Mapping (SLAM) has been a very important and widely studied problem, especially in the field of robotics. The modern state of the art SLAM methods largely do not incorporate deep learning methods, even though many breakthroughs and improvements have been made in the field of deep learning in the past several years, especially when applied to the field of computer vision.*

*In this paper, we describe and reimplement DeepSLAM, an end-to-end deep learning based SLAM method. DeepSLAM is comprised of three major components: MappingNet, TrackingNet, and LoopNet. MappingNet is an encoder-decoder network that, given an RGB image, predicts the depth of each pixel, producing a RGBD image. TrackingNet is a RNN that computes the translation between two consecutive frames. LoopNet is a feature extraction module that can detect when an agent revisits a location. Together, these modules form a complete SLAM method.*

*In this re-implementation, we've made a few modifications to the original DeepSLAM method. Notably, we don't implement MappingNet, and we try different architectures and training setups for TrackingNet and LoopNet than in the original paper. Ultimately, after developing TrackingNet and LoopNet, we were unable to successfully train TrackingNet.*

## 1. Introduction

In recent years, the advances in the fields of computer vision and deep learning have been applied to numerous fields, often touting unparalleled accuracy when compared to traditional methods. Within the last decade, improvements in the GPUs used to train neural networks, architectural improvements in neural networks, and modern frameworks that allow for rapid model development have all contributed to the incredible performance of modern deep learning methods.

However, despite all these advances, the Simultaneous Localization And Mapping (SLAM) problem has been a notable holdout. SLAM is an important method in the fields of robotics and augmented reality, due to the importance of knowing where an agent is in relation to its environment. In these applications, it's important that the SLAM method is robust to noise, can run at high speeds in real-time, and can successfully run for long periods of time. Due to these difficult requirements, many state of the art SLAM methods [14], [10], [4] don't make use of deep learning in their methods. Instead, these methods prefer traditional methods based on sensor fusion, traditional feature extraction from images [15], and nonlinear optimization methods.

In this paper, we are re-implementing the majority of a recent deep learning SLAM method, DeepSLAM [12]. DeepSLAM boasts several advantages compared to other deep learning SLAM methods. All of DeepSLAM's training can be done in an unsupervised manner, negating the need for expensive data labelling. In turn, this reduced cost greatly increases the size of the dataset that can be used in its training. Additionally, DeepSLAM's training is based entirely on stereoscopic image data, removing the need for a LiDAR sensor on the agent. (However, note that we are implementing a modified version of DeepSLAM in this paper, such that RGBD data will be used, in order to simplify the training process). At test time, DeepSLAM can be run with only a single camera, further reducing the cost of implementing it on an autonomous vehicle.

In this paper, we will discuss related methods in Section 2, fully describe the methods we used in Section 3, and give our results after re-implementing DeepSLAM in section 4.

## 2. Related Work

The SLAM technique has been used for several years for areas of Robotics and Augmented Reality where it is necessary to know the position of the agent and its environment. The first works with this technique used statistical models to infer pose and map, such as EKF-SLAM [19] which uses Kalman filters, Fast-SLAM [13] which uses Monte Carlo methods, or Graph-SLAM [21], which uses graph theory techniques.

However, in recent years different Deep Learning techniques have been incorporated into the SLAM pipeline due to the advances they have obtained in vision and long-term

memory tasks. CNNs have been implemented to predict pose in a supervised manner, such as PoseNET [9] which predicts 6-DOF camera pose with a single RGB image. This model has been implemented with other architectures achieving several improvements [22][3]. On the other hand, egocentric camera motion has also been predicted with neural networks [5]. Unsupervised learning techniques have also been used to estimate the camera pose [25].

Additionally, end-to-end deep learning SLAM methods have also been developed. MapNet [7] presents a method for localizing building a 2.5D map from point cloud readings, but is limited to only reasoning about flat environments. Prior to DeepSLAM, DeepVO [23] is perhaps the most robust end-to-end deep learning based approach, and can be seen as a direct precursor to DeepSLAM. However, DeepVO isn't quite a complete SLAM method, as it only performs localization, and does not include many modules, such as a loop closure detector, that have become standard in modern state of the art SLAM methods. DeepVO essentially lays the groundwork for DeepSLAM's TrackingNet, though DeepSLAM expands on this work by also including LoopNet and MappingNet. LIFT-SLAM [2], published following DeepSLAM, has further extended the methods of DeepVO and DeepSLAM in several ways, including using LIFT [24], a more robust feature extractor.

## 3. Methods

The original DeepSLAM paper is comprised of 3 main modules: MappingNet, TrackingNet, and LoopNet.

MappingNet is an autoencoder that, given the two stereoscopic RGB images at training time, produces two RGBD images. At test time, MappingNet is only given a single image to predict RGBD images from, and as such is a monocular depth estimator. In our paper, we focus on localization rather than mapping, so we omit MappingNet altogether.

The TrackingNet architecture is composed of two modules, an encoder and a decoder. The encoder is a convolutional neural network. Although in the original paper the authors use a VGG network from scratch, we decided to adopt the MobileNet v2 model [17], since it has a much smaller number of parameters for the same accuracy, and it is also cheaper to train. The output of this module is a vector of embeddings that are then decoded by the decoder. The decoder is a recurrent neural network, particularly an LSTM. This module receives the encoder's embeddings and the previous state in the temporal sequence of frames. The output of this module is the prediction of the 6DoF taken by the agent in each frame of the sequence.

Finally, LoopNet is a loop closure detection module. Loop closure detection is the problem of determining when an agent is in a position that it has already been in before, and using this knowledge to refine its estimates of its trajectory. For example, if a car stops at the same red light twice at $t_1$ and $t_2$, it's possible that sensor errors, odometry drift, or other errors building up in the TrackingNet has caused the agent to predict that its position at time $t_1$ is different from its position at time $t_2$. Situations like this are where a loop closure detection module can help improve the accuracy of a SLAM method. If the loop closure detection method can visually detect that the car has revisited the same red light, it can constrain the localization estimates to ensure that the position estimates at $t_1$ and $t_2$ are the same.

LoopNet is simply the convolutional layers of InceptionNet V3 [20] pre-trained on ImageNet [16]. Using InceptionNet, LoopNet extracts the 2048-dimensional feature vector from each frame of the data at test-time. Given two feature vectors, $f_i$ and $f_j$, their cosine distance $d_{ij}$ can be calculated as $d_{ij} = \frac{f_i \cdot f_j}{\|f_i\|\|f_j\|}$. Then a thresholding hyperparameter $\delta$ can be defined, such that two frames $i$ and $j$ of the video are said to be the same location if $d_{ij} < \delta$. Any pair $(i, j)$ satisfying this condition are then be passed back to TrackingNet in order to produce an output $(i, j, T_{ij})$, where $T_{ij}$ is TrackingNet's predicted transformation between frames $i$ and $j$.

Now, by simply passing a list of video frames $[I_1, I_2, ..., I_N]$ through TrackingNet, a list of predicted positions $[P_1, P_2, ...P_N]$ can be produced, where each $P_I$ is the position of the agent at frame $i$. Then, LoopNet can be used to find corrections $C_1, C_2, ..., C_K$, where each $C_a$ is a tuple $(i, j, T_{ij})$. These positions and corrections can be passed to a graph optimization library, such as g2o [11].

### 3.1. Training and Losses

The unsupervised training process is described in Figure 1, where we take a batch of frames captured by the camera and, with them, estimate the successive camera positions for each instant using the TrackingNet. Taking the color frames, the estimates of the camera positions and the depth frames, the loss (green arrows) can be calculated, and the be used for backpropagate the gradients (purple arrows). For our implementation, we used two losses:

- PCLoss (Photometric Consistency), which consists of taking each color frame and synthesizing (with the help of an STN [8]) the previous and next frame that should be seen by the camera according to:

$$p_{k+1} = K\hat{T}_{k,k+1}D_kK^{-1}p_k \tag{1}$$

where $p$ is the coordinate of a pixel, $K$ is the intrinsic matrix, $\hat{T}_{k,k+1}$ the estimated change of coordinates of the camera and $D_k$ the depth value for that pixel. Then the dissimilarities between the real and synthesized frames are measured using structural similarity and norm-1-similarity, and the loss is summed accord-

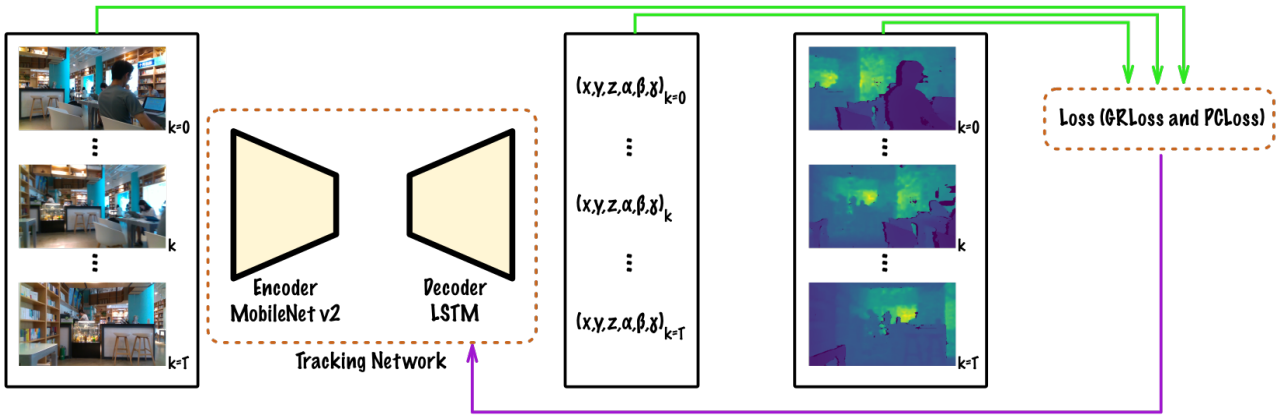


Figure 1. General scheme of the training environment using DeepSLAM. A sequence of $T$ color frames is passed through the TrackingNet and the 6DoF are estimated. Both sequences are used together with the depth frames to compute the overall loss and then train the network parameters. The forward flow is represented with green, while the backward is represented as purple.

ing to:

$$L^p_{k,k+1} = \sum (\lambda_s f_s(I_k, I'_k) + (1 - \lambda_s)||E^k_p||_1) \quad (2)$$

$$L^p_{k+1,k} = \sum (\lambda_s f_s(I_{k+1}, I'_{k+1}) + (1 - \lambda_s)||E^{k+1}_p||_1) \quad (3)$$

where $f_s(\cdot) = (1 - \text{SSIM}(\cdot))/2$ measures the structural similarity, $\lambda_s$ a balancing parameter, $I$ and $I'$ are the real and synthesized images, and $E$ is the error or difference between them.

- GRLoss (Geometric Registration), which consists of taking each depth frame, converting it into a color "image" having as RGB channels the XYZ coordinates and synthesizing (also with an STN) the previous and next frame that the camera should see according to Equation 1. Then the dissimilarities between the real and synthesized frames are measured using norm-1-similarity, and the loss is summed similarly to PCLoss:

$$L^g_{k,k+1} = \sum ||E^k_g||_1 \quad (4)$$

$$L^g_{k+1,k} = \sum ||E^{k+1}_g||_1 \quad (5)$$

where $E$ is the error or difference between real and synthesized images.

The overall loss can be computed as a weighted sum of these losses since the order of magnitude of each one of them is dissimilar: PCLoss measures difference between images using the RGB channels (values in $[0, 1]$), while GRLoss uses XYZ channels (values in mm). The weights for the sum are set as hyperparameters.

The authors of DeepSLAM also use other losses for the estimation of the depth frames, which is not necessary for our implementation because they are already given by the dataset. They also use outlier rejection to mask moving objects that may disturb the estimates, since outdoor datasets present many moving objects such as cars or pedestrians; however, we don't implement it here because indoor objects are generally static.

Table 1. LORIS Dataset

| Name | Size (GB) | Setting |
|---|---|---|
| Cafe | 7.0 | Indoor |
| Market | 39.0 | Indoor |
| Home | 18.0 | Indoor |
| Corridor | 31.0 | Indoor |
| Office | 9.9 | Indoor |
| Total | 104.9 | |

Table 2. Fixed Parameters

| Param | Value | Description |
|---|---|---|
| Batch Size | 96 | Images processed per batch |
| Train Split | 0.85 | Percent of Samples for training |
| Validation Split | 0.05 | Percent of Samples for validation |
| Test Split | 0.15 | Percent of Samples for test |
| Decoder LR | 1e-3 | Spatial Transformer decoder LR |
| Encoder LR | 1e-4 | Spatial Transformer encoder LR |

## 4. Evaluation

All development and testing was conducted using the OpenLORIS[18] dataset. OpenLORIS is very similar to the KITTI [6] and RobotCar [1] used in the original DeepSlam paper, but with a more manageable size. Another key difference is data was captured from robot mounted cameras in indoor environments. Additional dataset details are provided in table 1.

The majority of testing was performed on High-Ram

Google Colab Pro instances utilizing P100 or V100 GPUs. The code primarily uses PyTorch and is also configured to run locally on CPU or GPUs.

Training configurations are characterized by 13 parameters. Of those the 6 specified in table 2 were held constant. The 7 variable parameters (Data, Epochs, Samples, MobileNet, Speed, LR, and Lambda n) along with training time and test loss are detailed in table 3. MobileNet specifies whether the encoder CNN in tracking net should be retrained or use a pretrained network. Speed accelerates the robots movement through a scene by skipping frames - this is useful in environments where the robot moves very slowly. Lambda n gives controls the weighting of PCLoss and GRLoss; the fixed value weights them about equally. Samples is the total number of images that will be split across train, test, and validation during training.

For all combinations of parameters tested, little to no model improvement occurred during training. Although we could not identify any specific causes, there are multiple potential explanations. The DeepSlam authors had access to an NVIDIA DGX-1 with P100 GPUs, trained for longer, and trained with more data. Being limited to the storage, time, and compute resources of Colab Pro may have prevented successful training. Another concern is OpenLORIS captured indoor environments, while most KITTI and RobotCar sequences are outdoors. It is also possible our implementation was flawed. The original paper provided no code and very few model details. While all of our code runs there is significant room for bugs across the multiple networks and thousands of lines of code. Due to the poor performance of tracking net loop net could not be tested, because it requires the output from TrackingNet. Training and validation loss curves for all runs are included in the appendix.

Also, our DeepSLAM implementation was a simplified version of the original model, in which the authors added the outlier rejection. This module could help improve the architecture because it masks the pixels corresponding to moving objects in a given frame. In addition, the decoder module of our TrackingNet had a single layer, which could have prevented the network to learn complex temporal relationships between objects, causing a poor performance in the pose predictions.

## References


[1] 1 year, 1000 km: The Oxford RobotCar dataset - Will Maddern, Geoffrey Pascoe, Chris Linegar, Paul Newman, 2017. 3

[2] Hudson M. S. Bruno and Esther L. Colombini. Lift-slam: a deep-learning feature-based monocular visual slam method, 2021. 2

[3] Ronald Clark, Sen Wang, Andrew Markham, Niki Trigoni, and Hongkai Wen. Vidloc: A deep spatio-temporal model for 6-dof video-clip relocalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6856–6864, 2017. 2

[4] Alejo Concha and Javier Civera. Dpptam: Dense piecewise planar tracking and mapping from a monocular sequence. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5686–5693, 2015. 1

[5] Gabriele Costante, Michele Mancini, Paolo Valigi, and Thomas A Ciarfuglia. Exploring representation learning with cnns for frame-to-frame ego-motion estimation. *IEEE robotics and automation letters*, 1(1):18–25, 2015. 2

[6] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, June 2012. ISSN: 1063-6919. 3

[7] Joao F. Henriques and Andrea Vedaldi. Mapnet: An allocentric spatial memory for mapping environments. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8476–8484, 2018. 2

[8] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *arXiv preprint arXiv:1506.02025*, 2015. 2

[9] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015. 2

[10] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011. 1

[11] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011. 2

[12] R. Li, S. Wang, and D. Gu. DeepSLAM: A Robust Monocular SLAM System With Unsupervised Deep Learning. *IEEE Transactions on Industrial Electronics*, 68(4):3577–3587, Apr. 2021. Conference Name: IEEE Transactions on Industrial Electronics. 1

[13] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002. 1

[14] Raul Mur-Artal and Juan D. Tardos. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, Oct 2017. 1

[15] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 430–443, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. 1

[16] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy,

Table 3. Param Combos

| Data | Epochs | Samples | MobileNet | Speed | LR | lambda_n | Test Loss | Figure | ˜Time (min) |
|---|---|---|---|---|---|---|---|---|---|
| cafe1-1 | 50 | 10 | True | 1 | 0.001 | 3.000000e-07 | 0.000000 | 2 | 0.5 |
| cafe1-1 | 50 | 15 | True | 1 | 0.001 | 3.000000e-07 | 283.224335 | 3 | 1.0 |
| cafe1-1 | 50 | 50 | False | 1 | 0.001 | 3.000000e-07 | 480.484724 | 4 | 2.5 |
| cafe1-1 | 50 | 50 | False | 1 | 0.001 | 3.000000e-07 | 638.541221 | 5 | 26.5 |
| cafe1-1 | 50 | 50 | False | 2 | 0.001 | 3.000000e-07 | 512.208265 | 6 | 26.5 |
| cafe1-1 | 50 | 50 | False | 2 | 0.001 | 3.000000e-07 | 538.223005 | 7 | 2.5 |
| cafe1-1 | 50 | 50 | False | 10 | 0.001 | 3.000000e-07 | 550.656808 | 8 | 2.5 |
| cafe1-1 | 50 | 50 | True | 1 | 0.001 | 3.000000e-07 | 480.641671 | 9 | 2.5 |
| cafe1-1 | 50 | 50 | False | 1 | 0.005 | 3.000000e-07 | 480.540388 | 10 | 2.5 |
| cafe1-1 | 50 | 50 | False | 2 | 0.005 | 3.000000e-07 | 537.657854 | 11 | 2.5 |
| cafe1-1 | 50 | 50 | False | 10 | 0.005 | 3.000000e-07 | 541.739048 | 12 | 2.5 |
| cafe1-1 | 50 | 1024 | True | 1 | 0.001 | 3.000000e-07 | 512.749694 | 13 | 68.0 |
| cafe1-1 | 250 | 10 | True | 1 | 0.001 | 3.000000e-07 | 0.000000 | 14 | 3.3 |
| cafe1-1 | 250 | 15 | False | 1 | 0.001 | 3.000000e-07 | 283.224335 | 15 | 4.0 |
| cafe1-1 | 250 | 15 | True | 1 | 0.001 | 3.000000e-07 | 283.224335 | 16 | NaN |


Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 2

[17] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 2

[18] Qi She, Fan Feng, Xinyue Hao, Qihan Yang, Chuanlin Lan, Vincenzo Lomonaco, Xuesong Shi, Zhengwei Wang, Yao Guo, Yimin Zhang, Fei Qiao, and Rosa H. M. Chan. Openloris-object: A robotic vision dataset and benchmark for lifelong deep learning, 2020. 3

[19] Randall C Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68, 1986. 1

[20] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015. 2

[21] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006. 1

[22] Florian Walch, Caner Hazirbas, Laura Leal-Taixe, Torsten Sattler, Sebastian Hilsenbeck, and Daniel Cremers. Image-based localization using lstms for structured feature correlation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 627–637, 2017. 2

[23] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2043–2050. IEEE, 2017. 2

[24] Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. Lift: Learned invariant feature transform, 2016. 2

[25] Huangying Zhan, Ravi Garg, Chamara Saroj Weerasekera, Kejie Li, Harsh Agarwal, and Ian Reid. Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 340–349, 2018. 2

# 5. Appendix

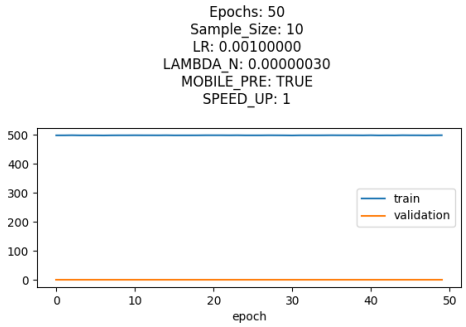


Figure 2.

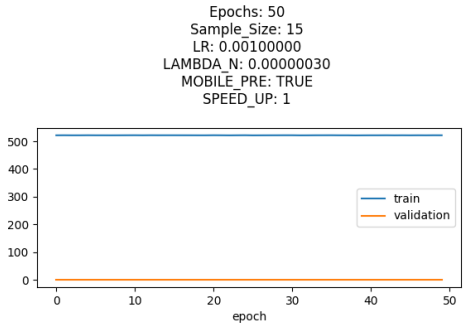


Figure 3.

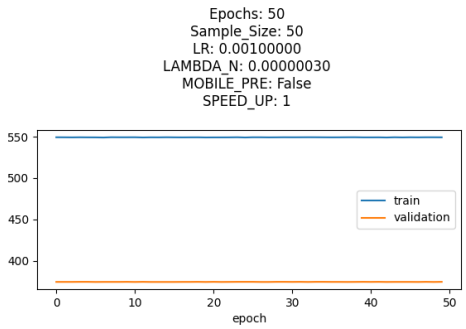


Figure 4.

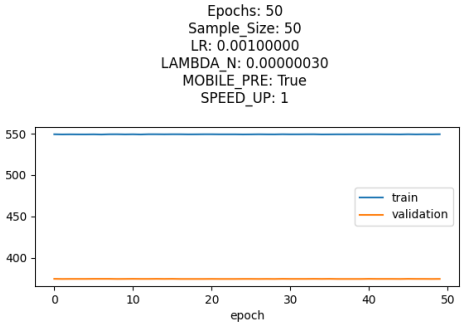


Figure 5.

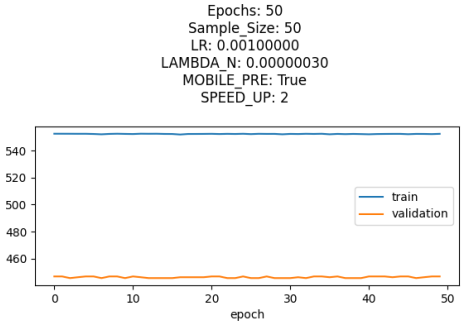


Figure 6.

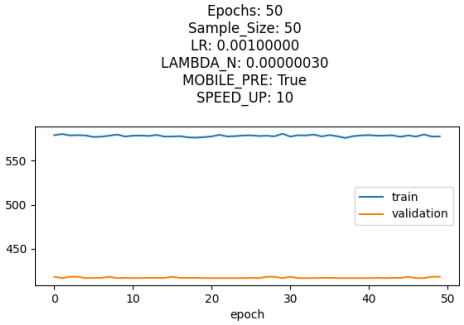


Figure 7.

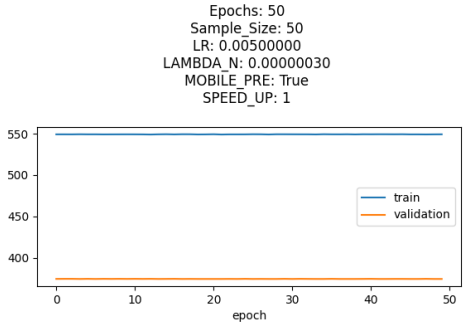


Figure 8.

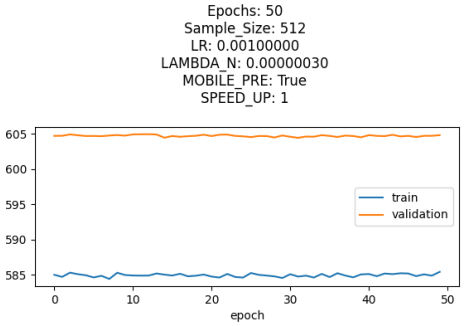


Figure 11.

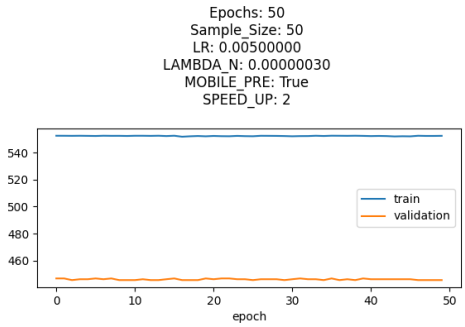


Figure 9.

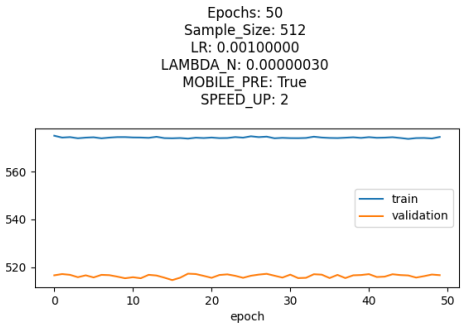


Figure 12.

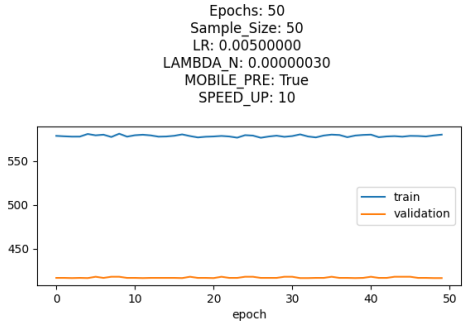


Figure 10.

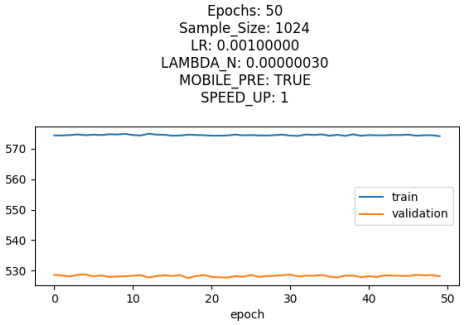


Figure 13.

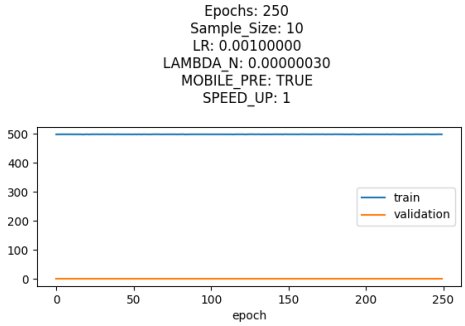


Figure 14.

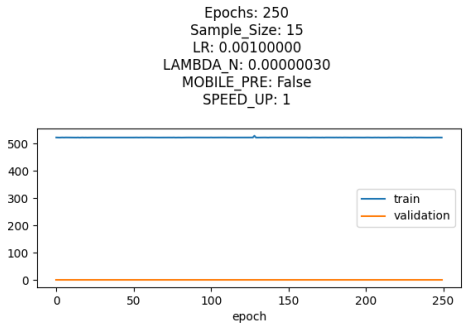


Figure 15.

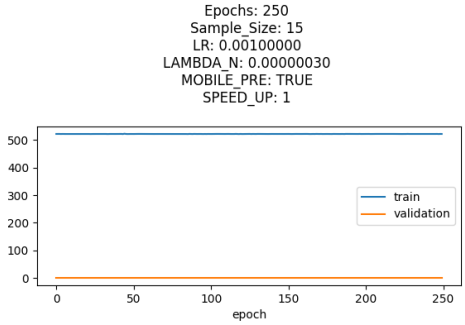


Figure 16.