LabMLISP: Lab Course Machine Learning in Signal Processing

Exercise VI: Hyperparameter Tuning

Prof. Veniamin Morgenshtern

Author: Angel Villar Corrales, Matthias Sonntag

The goal of this exercise is to perform experiments by varying the hyperparameters of the network in order to arrive at a set of optimal parameters for the network.

Exercise Materials

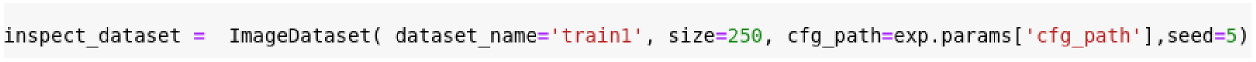
To copy the additional datasets used for this exercise, execute:

```
$ cd ~/labmlisp/data/input_data/
```

```
$ cp -r ~/SHARED/DATA/mlisp-lab/data/input_data/CARLA* .
```

Data setup

In order to perform the training experiments, you will be using simulated images. You would first need to generate 500 images from the simulator from exercise 2. You may generate these images as multiple smaller datasets (for e.g., 2 datasets of size 250 each) by providing different values as the seed argument. You are also free to make changes in the **sim_config.json** file. Please use the **02-test-image-dataset.ipynb** notebook to generate each dataset, so that you can generate each dataset individually after modifying the **sim_config.json** file each time.



```
inspect_dataset =  ImageDataset( dataset_name='train1', size=250, cfg_path=exp.params['cfg_path'],seed=5)
```

Figure 1: Code snippet from **02-test-image-dataset.ipynb** notebook demonstrating the creation of 'Sim_Lane_Train_s2' dataset with size 250 and seed 2.

In addition to the images generated by the simulator from exercise 2, we provide you with images generated using the CARLA simulator to use in your experiments. These datasets are named as CARLA_Train, CARLA_Test and CARLA_Predict.

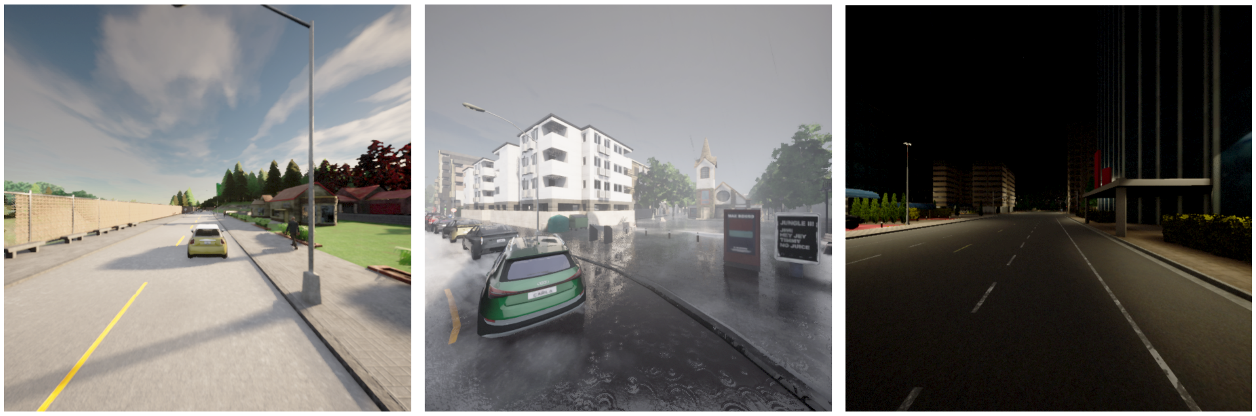
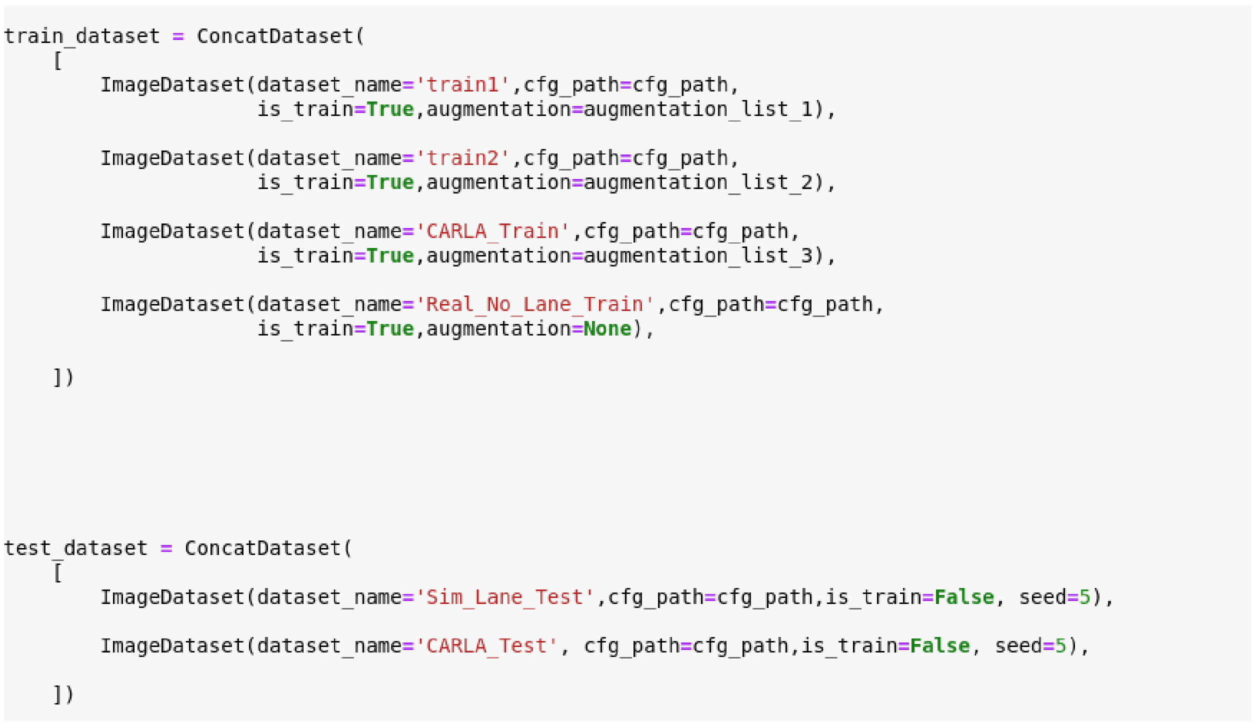


Figure 2: Examples of images generated by the CARLA simulator.

After all the datasets are created, you would only need to specify their names correctly in the **05-train.ipynb** notebook.



```
train_dataset = ConcatDataset(
    [
        ImageDataset(dataset_name='train1',cfg_path=cfg_path,
                     is_train=True,augmentation=augmentation_list_1),

        ImageDataset(dataset_name='train2',cfg_path=cfg_path,
                     is_train=True,augmentation=augmentation_list_2),

        ImageDataset(dataset_name='CARLA_Train',cfg_path=cfg_path,
                     is_train=True,augmentation=augmentation_list_3),

        ImageDataset(dataset_name='Real_No_Lane_Train',cfg_path=cfg_path,
                     is_train=True,augmentation=None),

    ])




test_dataset = ConcatDataset(
    [
        ImageDataset(dataset_name='Sim_Lane_Test',cfg_path=cfg_path,is_train=False, seed=5),

        ImageDataset(dataset_name='CARLA_Test', cfg_path=cfg_path,is_train=False, seed=5),

    ])
```

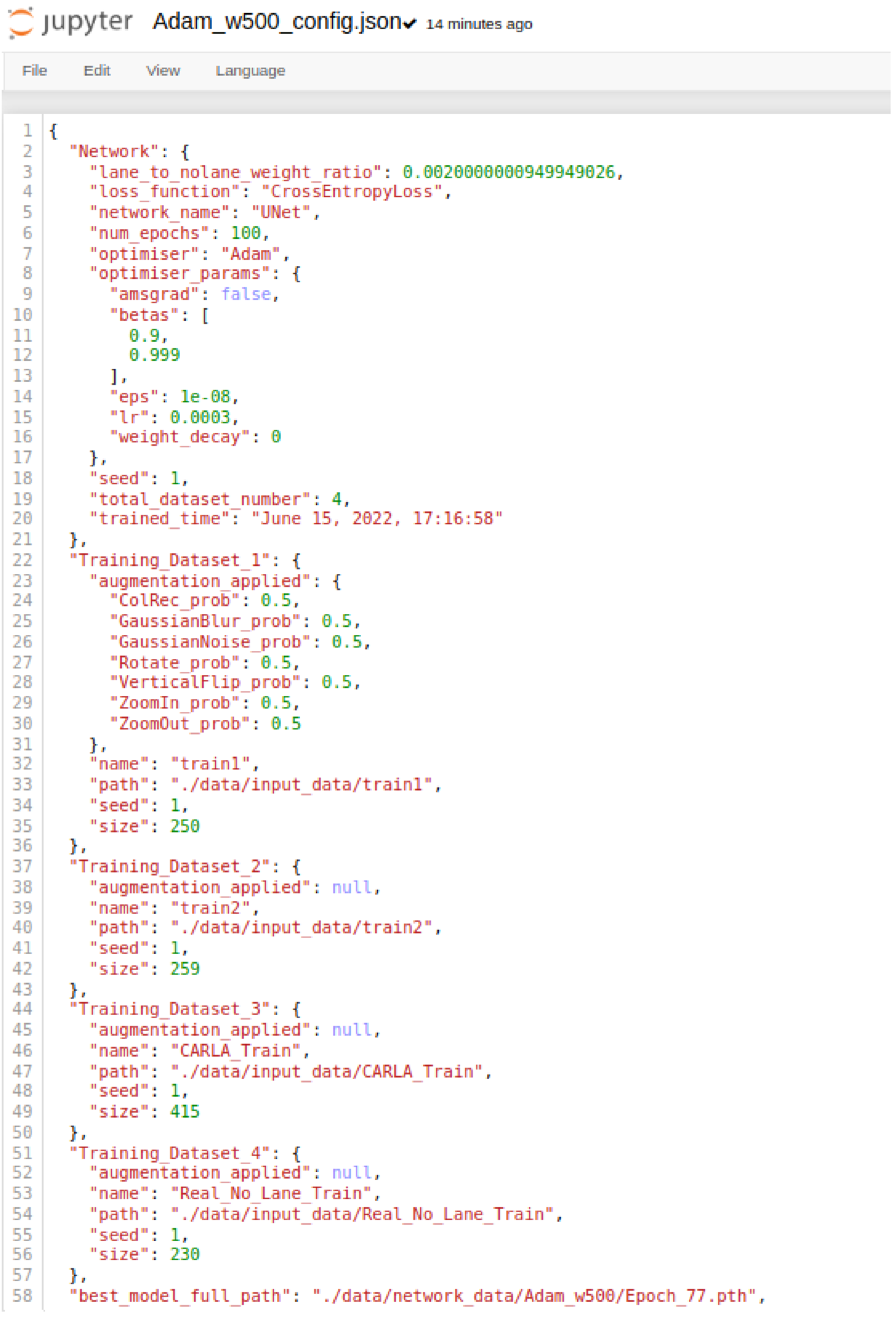
Figure 3: Code snippet from **05-train.ipynb** notebook demonstrating the concatention of multiple datasets to create a larger, more diverse train dataset. You could also use different augmentations for each dataset

Experiment setup

You will be conducting multiple experiments in this task. Use appropriate names of each experiment such that the name would help you identify how the network in that experiment was setup. If you use the same experiment name, the previous experiment will get overwritten!

In engineering in general (and Machine Learning in particular), it is crucial to document the results to be able to compare different experiments and to reproduce research. Create an excel file named **experiment-results.csv** to document details of every experiment such as experiment name, training parameters that were relevant/changed, test accuracy, test lane F1 score and other observations that you may have from your experiment. Optional: Since ground truth labels are not available for the data in the **06-predict.ipynb** notebook, you could give a your own rating based on the visual inspection of the results in the notebook.

After you train each experiment, along with the model weights, a config file is also generated in the **.data/network_data**. For example, if your experiment name is 'Adam_w500', the config file can be found at the path: **./data/network_data/Adam_w500/Adam_w500_config.json**. Note: The config file setup is NOT a feature of Pytorch. The code for this was already included along with the skeleton code that was provided to you for the previous lab tasks. The config file is meant to help you document your experiment results, in case you forget what parameters you had used for training.



```
 1 {
 2   "Network": {
 3     "lane_to_nolane_weight_ratio": 0.0020000000949949026,
 4     "loss_function": "CrossEntropyLoss",
 5     "network_name": "UNet",
 6     "num_epochs": 100,
 7     "optimiser": "Adam",
 8     "optimiser_params": {
 9       "amsgrad": false,
10       "betas": [
11         0.9,
12         0.999
13       ],
14       "eps": 1e-08,
15       "lr": 0.0003,
16       "weight_decay": 0
17     },
18     "seed": 1,
19     "total_dataset_number": 4,
20     "trained_time": "June 15, 2022, 17:16:58"
21   },
22   "Training_Dataset_1": {
23     "augmentation_applied": {
24       "ColRec_prob": 0.5,
25       "GaussianBlur_prob": 0.5,
26       "GaussianNoise_prob": 0.5,
27       "Rotate_prob": 0.5,
28       "VerticalFlip_prob": 0.5,
29       "ZoomIn_prob": 0.5,
30       "ZoomOut_prob": 0.5
31     },
32     "name": "train1",
33     "path": "./data/input_data/train1",
34     "seed": 1,
35     "size": 250
36   },
37   "Training_Dataset_2": {
38     "augmentation_applied": null,
39     "name": "train2",
40     "path": "./data/input_data/train2",
41     "seed": 1,
42     "size": 259
43   },
44   "Training_Dataset_3": {
45     "augmentation_applied": null,
46     "name": "CARLA_Train",
47     "path": "./data/input_data/CARLA_Train",
48     "seed": 1,
49     "size": 415
50   },
51   "Training_Dataset_4": {
52     "augmentation_applied": null,
53     "name": "Real_No_Lane_Train",
54     "path": "./data/input_data/Real_No_Lane_Train",
55     "seed": 1,
56     "size": 230
57   },
58   "best_model_full_path": "./data/network_data/Adam_w500/Epoch_77.pth",
```

Figure 4: Example of the config file generated after the model training in the experiment named as "Adam_w500". The number shown in the weight ratio (line 3) is 1:500 represented as a floating point number. The config file is generated using parameters that you enter in the **05-train.ipynb** notebook, and when you don't enter any values, the default values will be used.

Baseline Network

The first task will be to train network with a baseline configuration. You can use the evaluation from this network as a benchmark for the remaining experiments. A sample baseline configuration is provided in Table 1. However, you are free to make any changes if required.

Parameter	**Value**
Optimizer	Adam
Learning Rate	0.0002
Augmentation	None
Training Epochs	100
Loss Function Weight Ratio	1:500

Table 1: Parameter configuration for the Baseline network

Experiments

As the total number of combinations of parameters is quite large and trying all of them would require a lot of time, we have limited the number of parameters that we will cover in this task.

- **Optimizers** – Experiment with any two optimizers available in Pytorch (for e.g., Adam and SGD Optimizer). Run each optimizer for 100 epochs with the default optimizer parameters. Observe the performance of the network using the test dataset F1 score during training and the visual outputs from the Predict notebook. Also observe how the value of loss function varies, using tensorboard. Note that some optimizers may reach converge to the best model parameters very quickly and you may not be required to train for 100 epochs.

- **Learning Rate** – Experiment with the learning rate by training the model. You may test the performance with learning rates like [0.001, 0.0001, 0.0001]. Based on the performance of these models, you can perform a more finer search around that point. For example, if 0.001 gave the best performing model in the previous search, then you could run a second round of search with values like [0.005, 0.0025, 0.00075, 0.0005]. Optional: You may also vary the other optimizer parameters if applicable.

- **Loss Function Weights** – As you may have noticed, in each image the number of pixels corresponding to lane is much less than number of pixels corresponding to non lane. This problem, referred as 'Class Imbalance', is quite common in segmentation tasks. The imbalance might lead to the network preferring to predicting most pixels as 'not lane' in order to achieve a lower loss. One of the ways to mitigate this problem is to use a weighted loss function, such that there is greater penalty when the network fails to detect a lane. We shall use higher weights to the pixels corresponding to lane in our loss function and thereby compensate the class imbalance. Vary the weights using code in the cell 7 of the *05-train.ipynb* notebook and find the appropriate weights for the lane in a similar manner as the learning rate search.

- **Augmentations** – Train the network with different data augmentations and combinations of data augmentations. You may also change the probabilities of each augmentation as required. Which augmentations proved to be most effective?

Based the above experiments, you may further fine tune these parameters in order to optimize them for your network.

Submission

- As you perform these experiments, make a note of the best performing network by visually inspecting the results from the **06-predict.ipynb** notebook, especially the *Real_Lane_Predict* dataset with real road lane images.
- Show the results from your best network to one of the tutors.
- Upload the excel document **experiment-results.csv** with your experiment results and the **06-predict.ipynb** notebook as a html file to StudOn.