ETH zürich

Deep Learning for Autonomous Driving, Spring 2021

Multi-Task Learning for Semantics and Depth

——Project Report

Yifei Liu (Team 51)
yifei.liu@uzh.ch

May 14th, 2021

1 Problem1: Joint Architecture

1.1 Hyper-parameter tuning

—— In this task, we are already provided a baseline code that can function out of the box. And the aim of this task is to find as best hyper-parameters as possible for the baseline model.

The candidate hyper-parameters which should be tuned are specified to: (1)*optimizer*, (2)*learning rate*, (3)*batch size*, and (4)*task weights*. In my experiments, I first explore the first three hyper-parameters to pick out a best and stable combination, and then go ahead with the forth hyper-parameter. The running results are in Figure 1, Figure 2 and Table 1.

My procedure to explore hyper-parameters are:

1. fix the *optimizer* to be *sgd*.
2. change the learning rates logarithmically.
3. change batch size and for every batch size do step 2 again.
4. change optimizer to be *adam* and do step 2 and 3 again.
5. fix a best combination for the previous three hyper-parameters, then change task weights

Notice that number of epochs always equals four times the batch size, which ensures a constant number of steps for each training, making the results comparable.

Analyzing the results:

- The default learning rate 1×10^{-2} is not optimal for *sgd* for this task (Figure 1). The best learning rate for *sgd* is around 1×10^{-1}, while the best learning rate for *adam* is around 2×10^{-4}. By the way, if the learning rate deviates the best setting too much, the performance of the model drops down dramatically.
- Larger batch sizes are better for improving the model performance (Figure 1). But the training time and cost also increase almost linearly, so it is a trade-off between cost and accuracy.

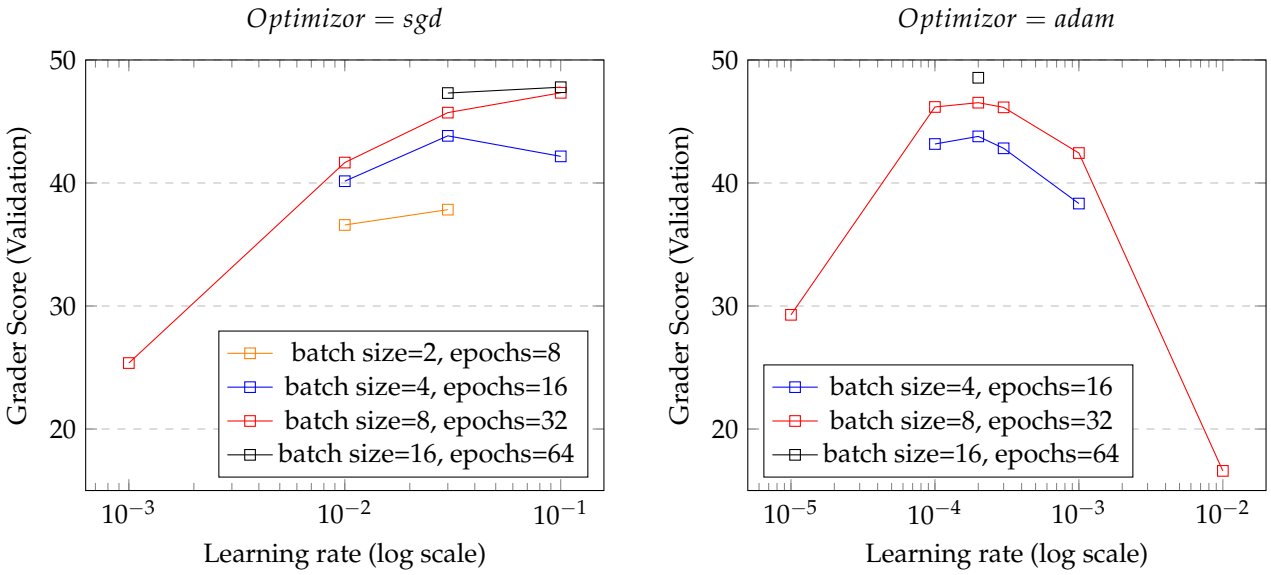

Figure 1: Grader score of *sgd* and *adam* with different learning rates and batch sizes

- For a specific setting, changing the loss weights only affects the performance very little (Figure 2). And the best performance is attained while loss weight of semantic segmentation = 0.7 (Figure 2).
- By trading off the budget and accuracy, **I decide to use the bold set of hyper-parameters in Table 1 as the general setting for all the following problems, unless otherwise stated**. Notice that given $batch\_size = 8$, "*sgd* with $lr = 0.1$" gets the the highest grader score among all runs. But it turns out to be unstable for training because the learning rate is such large that sometimes the loss would go infinity, causing the model not to converge. This is why I choose adam instead.
- The run that produces the best validation score is G51_0424-2257_adam_0.0002_batch_16_epoch_64_91977 with a validation score of 48.554 and a grader score of 48.304 in Codalab.
- The baseline model G51_0425-1704_weight_seg_0.7_adam_0.0002_batch_8_epoch_32_e6c97 for all following problems has a validation score of 46.807, and a Codalab score of 46.603.

Name	Optimizer	Learning rate	Batch size	Loss weight	Grader score(val)
G51_0424-2257_adam_0.0002_batch_16_epoch_64_91977	adam	0.0002	16	0.5	48.554
G51_0422-0009_sgd_0.1_batch_16_epoch_64_5f080	sgd	0.1	16	0.5	47.785
G51_0420-0442_sgd_0.1_batch_8_epoch_32_f2743	sgd	0.1	8	0.5	47.333
G51_0423-2230_sgd_0.03_batch_16_epoch_64_833d8	sgd	0.03	16	0.5	47.311
G51_0425-1704_weight_seg_0.7_adam_0.0002_batch_8_epoch_32_e6c97	**adam**	**0.0002**	**8**	**0.7**	**46.807**
G51_0423-2221_adam_0.0002_batch_8_epoch_32_52b42	adam	0.0002	8	0.5	46.529

Table 1: summary of (highest) different hyper-parameters in Task 1.1

1.2 Hardcoded hyperparameters

1.2.1 Initialization with ImageNet weights

Q: Is the encoder initialized with weights of a model trained on the ImageNet classification task?
A:No. In the *ModelDeepLabV3Plus* class the "pretrained" parameter is set by default to False.

Q:What is the effect of switching this option?

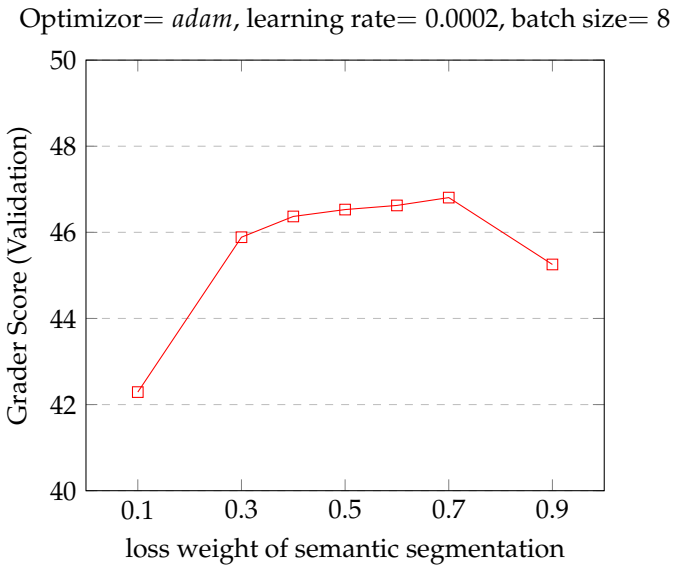

Figure 2: Influence of different loss weights (depth weight = 1− loss weight of semantic segmentation)

A:Using pretrained weights from ImageNet classification improves performance of the model (Table 2), since many lower features and general representations are shared by this ResNet34 encoder. And since the pretrained weights were trained on large amount of data, it would be helpful for extracting useful low-level features. **For all the following problems, I use pretrained=True as the general setting, unless otherwise stated.** The Codalab score of the pretrained model is 48.501.

Name	Pretrained	Grader score(val,Codalab)
G51_0426-2134_pretrained_True_weight_seg_0.7_adam_0.0002_6715e	**True**	**48.701, 48.501**
G51_0425-1704_weight_seg_0.7_adam_0.0002_batch_8_epoch_32_e6c97	False	46.807, 46.603

Table 2: Influence of using pretrained weights

1.2.2 Dilated convolutions

Q: Are dilated convolutions enabled in the provided code?
A: No. The dilation was prohibited by default due to the flag replace_stride_with_dilation=(False, False, False). The BasicBlock's architecture is Figure 3. The default encoder with flag=(False, False, False) is Figure 4.

Q: Set dilation flags to (False, False, True). Does the performance improve? If so, why?
A: Yes, the validation score increases from 48.701 to 59.691 (Table 3). The architecture of Encoder with dilation is in Figure 5. Comparing Figure 4 and Figure 5, we can see the differences of using dilation or not are in the last three basic blocks (blue blocks) and the resulting resolution. The dilation increases receptive field without losing resolution, so the following features preserve a higher resolution and more detailed semantic information, thus creating better performance.

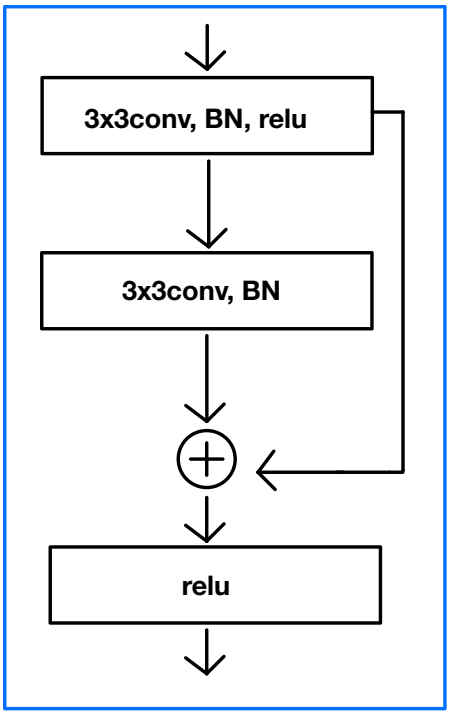

Figure 3: BasicBlockWithDilation. The dilation of the first 3×3conv is 1. The stride of the second 3×3conv is 1. If the stride of the first 3×3conv >1 or number of channels changes, the skip connection should be downsampled by a 1×1conv. The padding of the second 3×3conv = dilation, keeping the spatial resolution.

Name	Flag	Grader Score(val,Codalab)
G51_0426-2134_pretrained_True_weight_seg_0.7_adam_0.0002_6715e	(False,False,False)	48.701, 48.501
G51_0505-0106_FalseFalseTrue_adam_0.0002_e3376	**(False,False,True)**	**59.898, 59.816**

Table 3: Influence of enabling dilations in the encoder

1.3 ASPP and skip connections

The change of the original template code are *ASPPpart* class, *ASPP* class, *DecoderDeeplabV3p* class in *model_parts.py* and *ModelDeepLabV3Plus* class in *model_deeplab_v3_plus.py*. Figure 6 summarizes my implementation of the **ASPP** class, while Figure 7 summarizes the **DecoderDeeplabV3p** class. My implementation(architecture design, hyperparameter choice) is based on [2]. For example, number of 3×3 convolution layers in the decoder is 2, and number of channels in the skip connection is 48.
The ASPP and skip connection improve the performance a lot, increasing the validation score from 59.898 to 68.26, while the expected score of this model was 62.3 ±4.0 (Table 4)

Name	Dilation	ASPP and skip	Grader score(val,Codalab)
G51_0505-0106_FalseFalseTrue_adam_0.0002_e3376	✓		59.898, 59.816
G51_0506-2134_ASPP_with_skip_8494e	✓	✓	68.26, 68.296
Expected model performance	✓	✓	62.3 ± 4.0

Table 4: Performance of ASPP with skip models

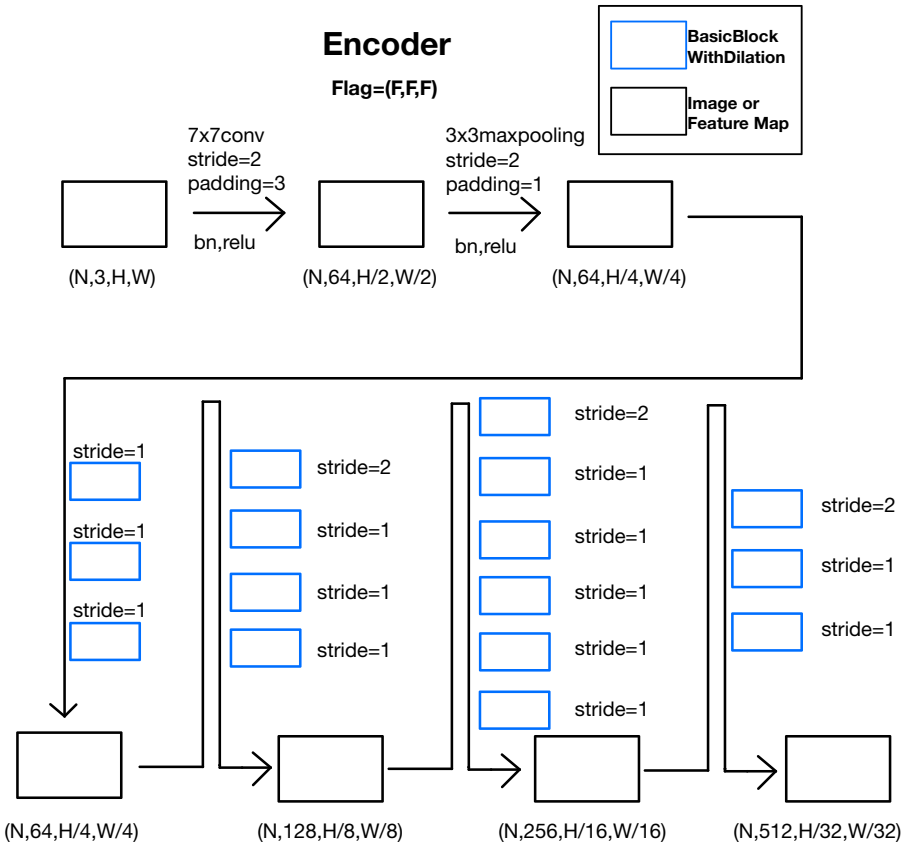

Figure 4: Encoder, flag=(F,F,F)

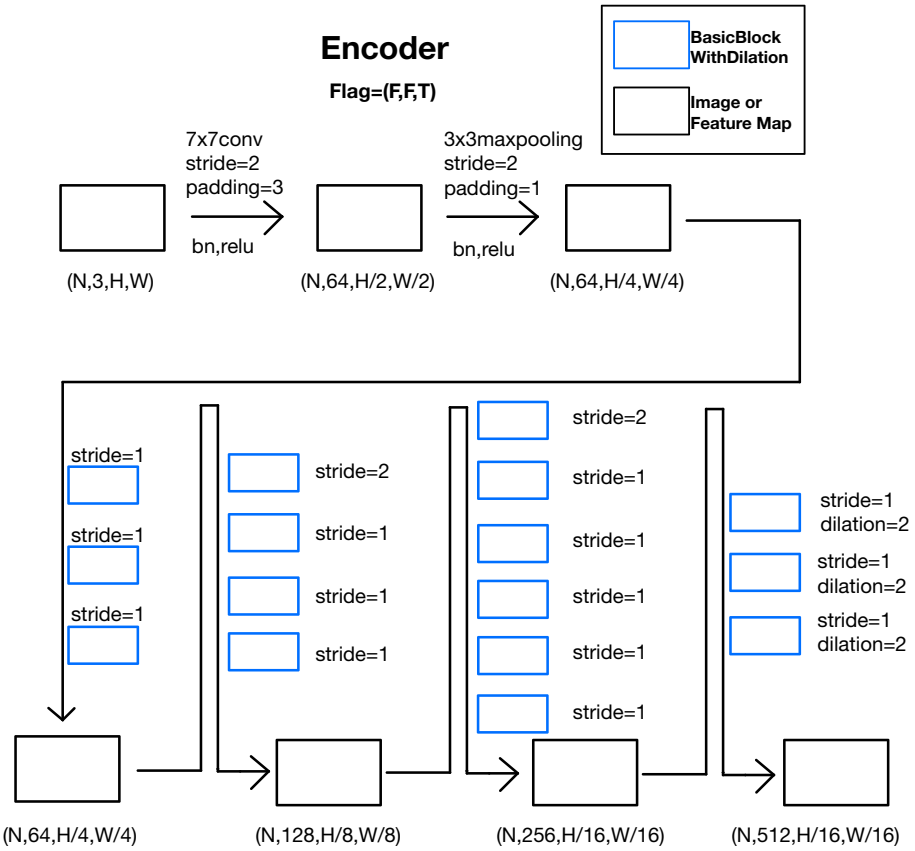

Figure 5: Encoder, flag=(F,F,T)

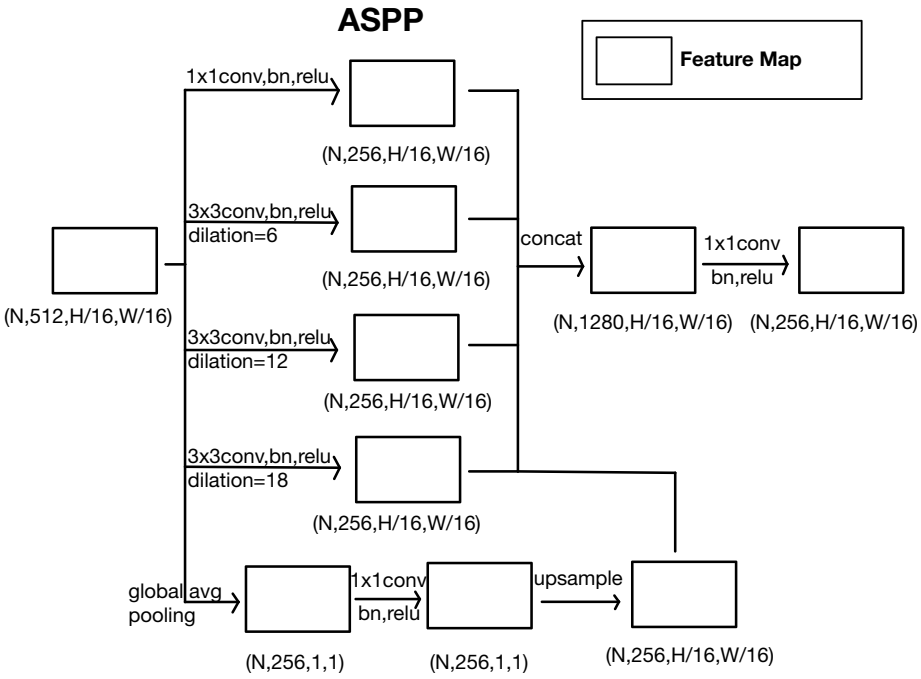

Figure 6: The ASPP architecture

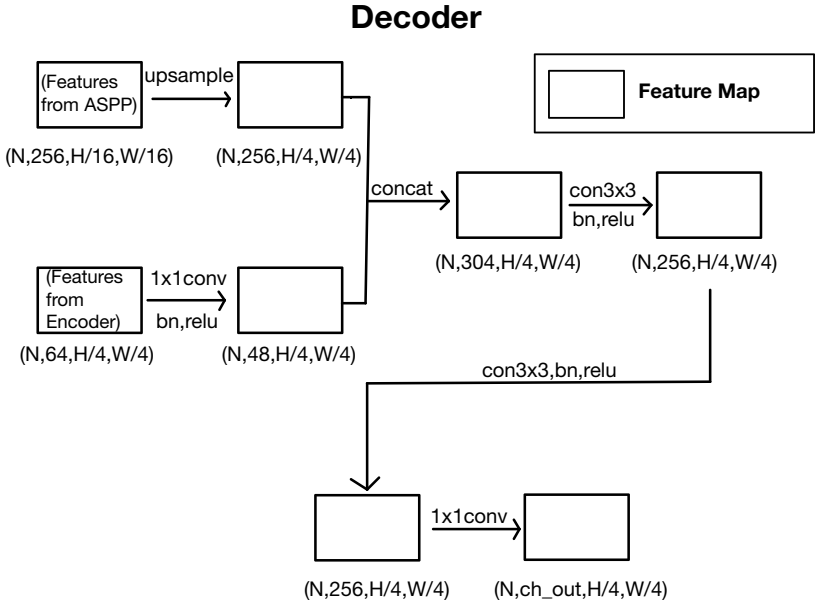

Figure 7: The Decoder architecture

2 Problem2: Branched Architecture

For the branched model, all the code in *model_parts.py* remain the same as in Problem 1.3. The only change of code is that I created a new source file called *branched_model.py*. And of course the *config.py* and *helpers.py* were changed accordingly. The implementation is straightforward and follows the guidance of the corresponding graph in the handout. The relevant code is in *branched_model.py*.

The branched architecture improves the performance by about 1.5 grader score. See Table 5.

Name	Dilation	ASPP and skip	Branch	Grader score(val,Codalab)
G51_0505-0106_FalseFalseTrue_adam_0.0002_e3376	✓			59.898, 59.816
G51_0506-2134_ASPP_with_skip_8494e	✓	✓		68.26, 68.296
G51_0508-0928_Branched_d322f	✓	✓	✓	69.709, 69.771
Expected model performance	✓	✓	✓	65.3 ± 4.0

Table 5: Performance of the branched model

3 Problem3: Task Distillation

The relevant code of distillation model is in *distillation_model.py*. The implementation strictly follows the corresponding graph given in the handout. Note that for the first decoder for depth and semantic segmentation I use the same architecture as before(Figure 7), while for the last decoder for the depth and semantic segmentation I simply use two 3×3 convolution plus batch normalization and relu, followed by a 1×1 convolution which produces the final prediction at the scale of a quarter of the original resolution. The relevant code for the last decoder is *FinalDecoder* class in *model_parts.py*

I experimented the distillation model with intermediate supervision and without intermediate supervision, and found that the intermediate supervision actually decrease the performance a little (the 4^{th} row and the 5^{th} row in Table 6).

Name	Dilation	ASPP and skip	Branch	Distillation	Intermed.Supv	Grader score(val,Codalab)
G51_0505-0106_FalseFalseTrue_adam_0.0002_e3376	✓					59.898, 59.816
G51_0506-2134_ASPP_with_skip_8494e	✓	✓				68.26, 68.296
G51_0508-0928_Branched_d322f	✓	✓	✓			69.709, 69.771
G51_0510-1259_Distillation_db1a3	✓	✓	✓	✓		70.333, 70.512
G51_0514-1004_Distillation_Intermed_Supv_bed8b	✓	✓	✓	✓	✓	70.314, 70.497
Expected model performance	✓	✓	✓	✓	✓	67.3± 4.0

Table 6: Performance of the distillation model

4 Summary

Table 6 shows that the methods in this project (including dilation, ASPP, skip connection, branched architecture, multi-task distillation) all contribute to improving the performance of the model, just as shown in [1], [3], [4], and [5]. My best model's performance is 70.512, without using any method mentioned in Problem 4. Since Problem 4 is cancelled, my report ends here.

References

[1] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. "Rethinking Atrous Convolution for Semantic Image Segmentation." In: *CoRR* abs/1706.05587 (2017). arXiv: 1706.05587. URL: http://arxiv.org/abs/1706.05587.

[2] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation." In: *CoRR* abs/1802.02611 (2018). arXiv: 1802.02611. URL: http://arxiv.org/abs/1802.02611.

[3] Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. "Fast Scene Understanding for Autonomous Driving." In: *CoRR* abs/1708.02550 (2017). arXiv: 1708.02550. URL: http://arxiv.org/abs/1708.02550.

[4] Simon Vandenhende, Bert De Brabandere, and Luc Van Gool. "Branched Multi-Task Networks: Deciding What Layers To Share." In: *CoRR* abs/1904.02920 (2019). arXiv: 1904.02920. URL: http://arxiv.org/abs/1904.02920.

[5] Dan Xu, Wanli Ouyang, Xiaogang Wang, and Nicu Sebe. "PAD-Net: Multi-tasks Guided Prediction-and-Distillation Network for Simultaneous Depth Estimation and Scene Parsing." In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 675–684.