DLAD Project 2 Report

Kexin Shi, Zheng Shen

May 2021

1 Joint Architecture

1. Hyper-parameter tuning

   (a) *Optimizer* and *LR* choice

   We have tried different combinations of optimizer and learning rate shown in Fig.1. With a small learning rate, the optimizer converges very slowly or might not be converged within the given running epochs. As we increase learning rate, the training process might converge quickly, however, it may also result in heavy oscillation. The best result is achieved by using learning rate 0.0001 with *Adam* optimizer and shown in Table.1.

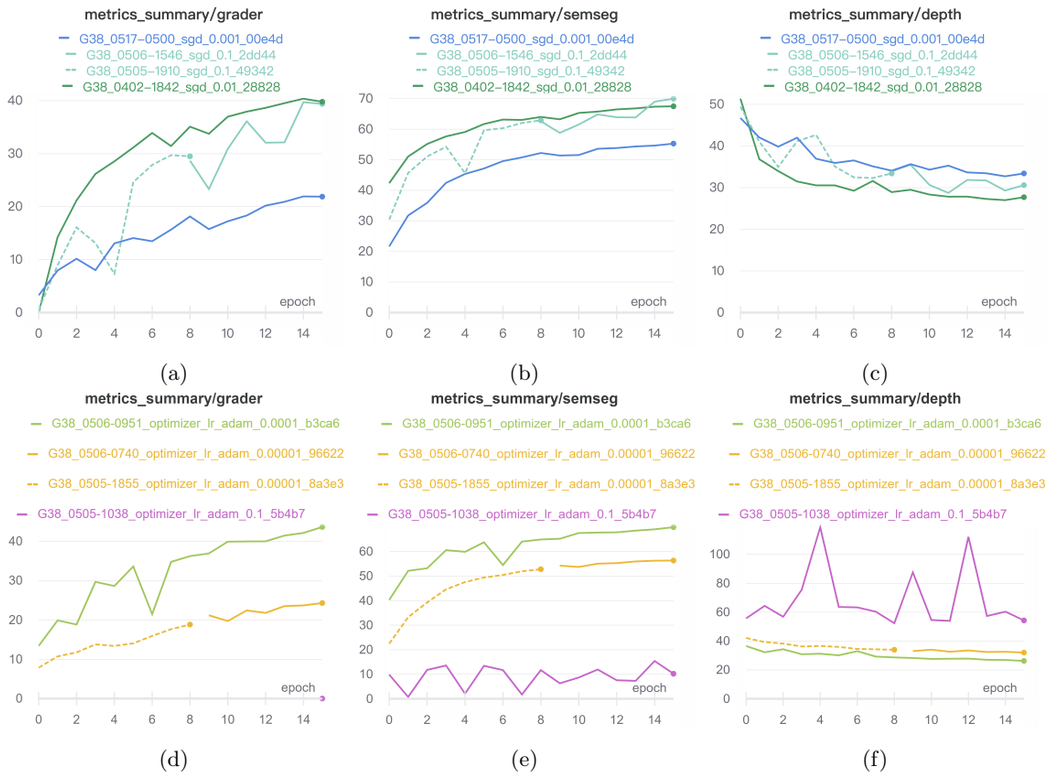


Figure 1: SGD and Adam optimizer with different learning rates

Table 1: Optimizer & Learning Rate Summary

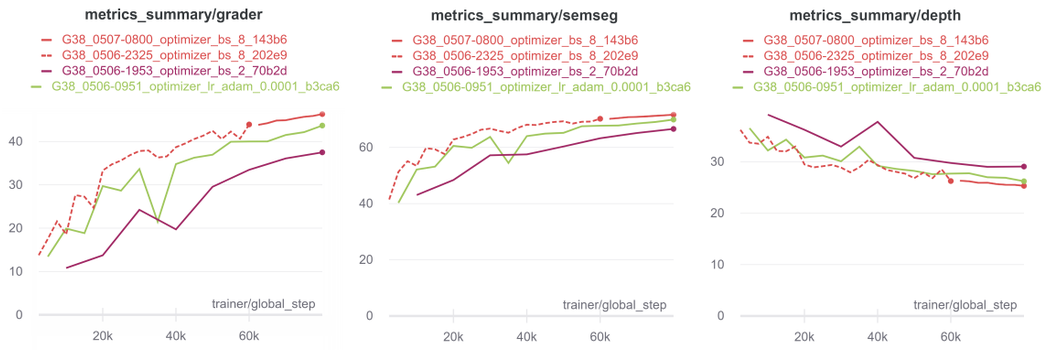
Optimizer	Learning rate	Grader	Semseg	Depth	Run
Adam	0.0001	43.686	69.919	26.333	G38_0506–0951_optimizer_lr_adam_0.0001_b3ca6

   (b) *Batch size*

   We tried different batch sizes: 2, 4, 8. As Fig.2 shows, the curve corresponding to the batch size of 8 is smoothier than the others. A larger batch size is preferred and performs better. The reason could be that a larger batch size gives a more accurate and stable gradient with optimizing the loss simultaneously over a larger set of images. However, the larger the batch size, the easier the training will crash. To train more stably, we use batch size of 4 in the following tasks although batch size of 8 gives better results currently. The best result is shown in Table 2.

Table 2: Batch Size Summary

Batch size	Epoch	Grader	Semseg	Depth	Run
8	32	46.336	71.672	25.336	G38_0507–0800_optimizer_bs_8_143b6

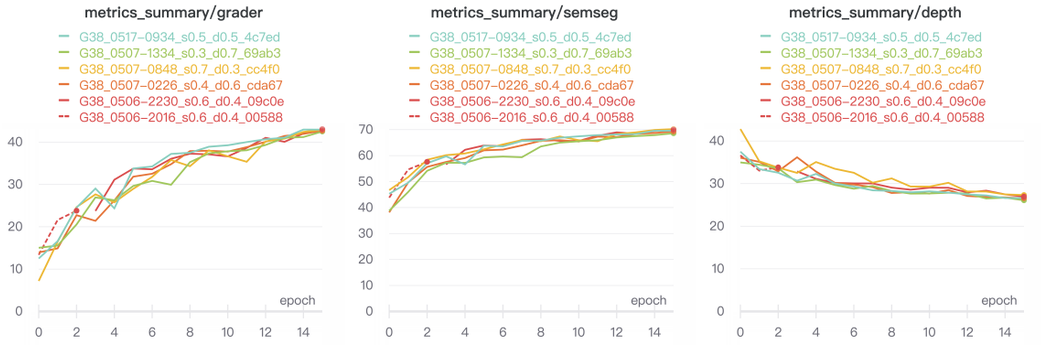


(a) (b) (c)

Figure 2: Comparison w.r.t. batch size

(c) *Task weighting*

We tried five different weight combinations shown in Fig.3 and Table 3. The grade is the highest when the two tasks are assigned with equal weights. From Table 3, we can observe that when we put more weights on semseg task and less weights on depth task, the performance of semseg task increases while the performance of depth task decreases. This is the essential reason why actually there is no significant difference between the overall scores. Considering the final grader is a combination of semseg and depth tasks, we need to make a trade-off between different tasks. Finally, we choose the equal weights for these two tasks to avoid one task overwhelms the other.



(a) (b) (c)

Figure 3: Comparison w.r.t. task weights

Table 3: Task Weight Summary

Semseg weight	Depth weight	Grader	Semseg	Depth	Run
0.3	0.7	42.52	68.61	26.09	G38_0507–1334_s0.3_d0.7_69ab3
0.4	0.6	42.73	69.302	26.571	G38_0507–0226_s0.4_d0.6_cda67
0.5	**0.5**	**43.686**	**69.919**	**26.333**	**G38_0517–0934_s0.5_d0.5_4c7ed**
0.6	0.4	43.008	70.035	26.947	G38_0506–2230_s0.6_d0.4_09c0e
0.7	0.3	42.917	70.229	27.312	G38_0507–0848_s0.7_d0.3_cc4f0

2. Hardcoded hyperparameters

   (a) Initialization with ImageNet weights

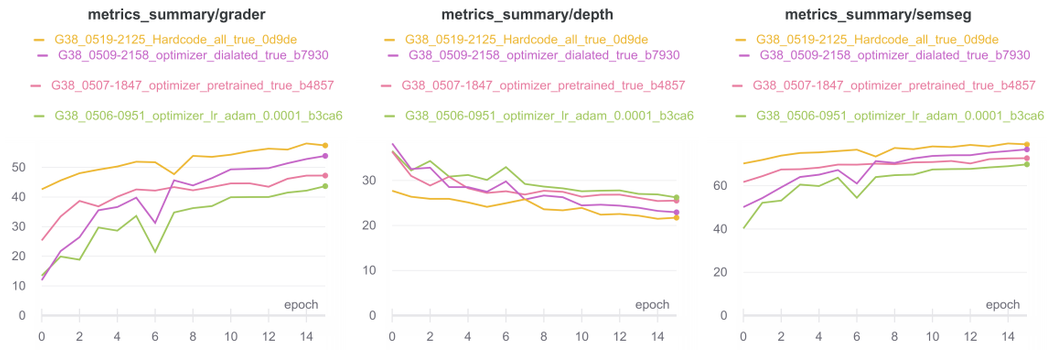
   At the beginning, the encoder network is not initialized with weights. After switching on initialization with ImageNet weights, the network will start from these pretrained weights instead of random distribution. As Fig.4 shows, pretrained weights provide a better starting point comparing to other runs, especially for semantic segmentation.

   (b) Dilated convolutions

   After setting dilation flags to (False, False, True), the fourth layer of encoder will be convolved with dilation. From the Fig.4 and Table 4, the performance with dilated convolutions is improved significantly based on the reason that the dilated convolutions provide exponential expansion of the receptive field without any loss of resolution information. Meanwhile, it do not increase the cost of computation and memory.

3. *ASPP and skip connections*

   In the following tasks, each run is set with best hyperparameters we found before: *Adam* optimizer, learning rate of 0.0001, batch size of 4, true flag for pretrained weights and (False, False, True) flags for dilated convolutions.



(a) (b) (c)

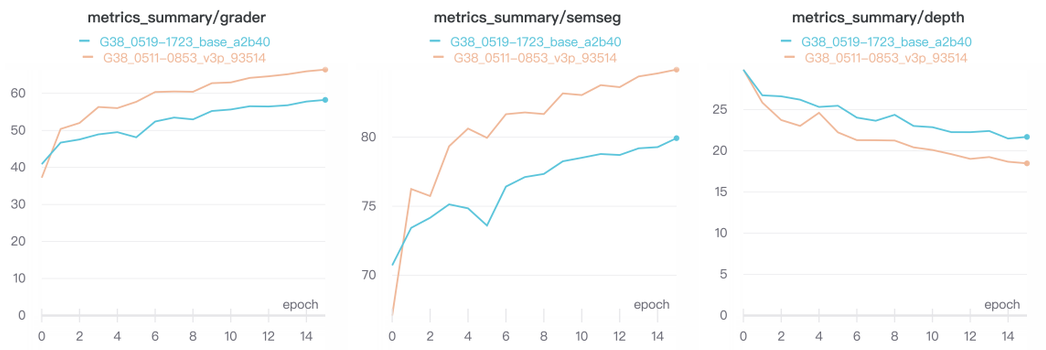
Figure 4: Comparison w.r.t. hardcoded hyperparameters

Table 4: Hardcoded Hyperparameters Summary

Pretrained	Dilated convolutions	Grader	Semseg	Depth	Run
Off	Off	43.686	69.919	26.333	G38_0506–0951_optimizer_lr_adam_0.0001_b3ca6
On	Off	47.258	72.780	25.522	G38_0507–1847_optimizer_pretrained_true_b4857
Off	On	53.911	76.845	26.233	G38_0509–2158_optimizer_dialated_true_b7930
On	On	57.438	79.176	21.738	G38_0519–2125_Hardcode_all_true_0d9de

For this part, code modification can be found in *mtl/models/model_parts.py*. With ASPP and skip connections, the performance is improved from 58.246 to 66.427.

Atrous Spatial Pyramid Pooling(ASPP) adopts parallel atrous convolution with different rates which is used to extract multi-scale contextual information. Skip connections allow decoder to combine low-level detailed information from DCNN with high-level semantic information from ASPP part. Before concatenating the upsampled high-level features from ASPP with the corresponding low-level features from the network backbone, we adopt one $[1 \times 1, 48]$ convolution on the low-level features for channel reduction, since the corresponding low-level features usually contain a large number of channels, which may outweigh the importance of the rich encoder features and make the training harder. After the concatenation, we adopt two $[3 \times 3, 256]$ convolutions to refine the features followed by another simple bilinear upsampling by a factor of 4, which is a simple yet effective decoder module.



(a) (b) (c)

Figure 5: Comparision of w/o and w/ ASPP and Skip Connection

Table 5: w/o and w/ ASPP and Skip Connection Summary

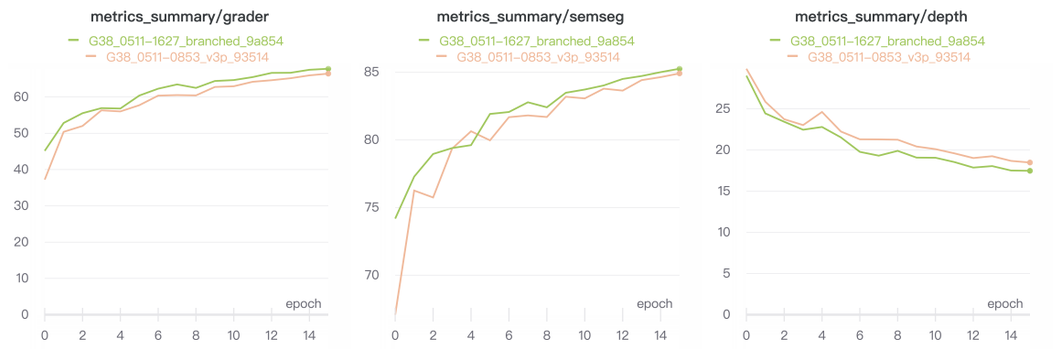
	Grader	Semseg	Depth	Run
w/o ASPP and Skip Connection	58.246	79.932	21.687	G38_0519–1723_base_a2b40
w/ ASPP and Skip Connection	66.427	84.902	18.475	G38_0511–0853_v3p_93514

2 Branched Architecture

Code for this architecture can be found in *mtl/models/model_branched.py*. Add a command line with `--model_name 'branched'` in *aws_train.sh* to use this model. Compared with the joint architecture, the branched model improves the performance from 66.427 to 67.778.

This architecture still shares the same encoder to extract information from pictures, as semantic segmentation and depth estimation are closely related. The main difference between it and joint architecture is they use separated ASPP part and decoder. This branched design allows the model to train task-specific decoders,

which improves the score of each individual task. However, the model size and computation cost will increase because the model have one more ASPP part and one more decoder.



(a) (b) (c)

Figure 6: Comparision of Joint Architecture and Branched Architecture

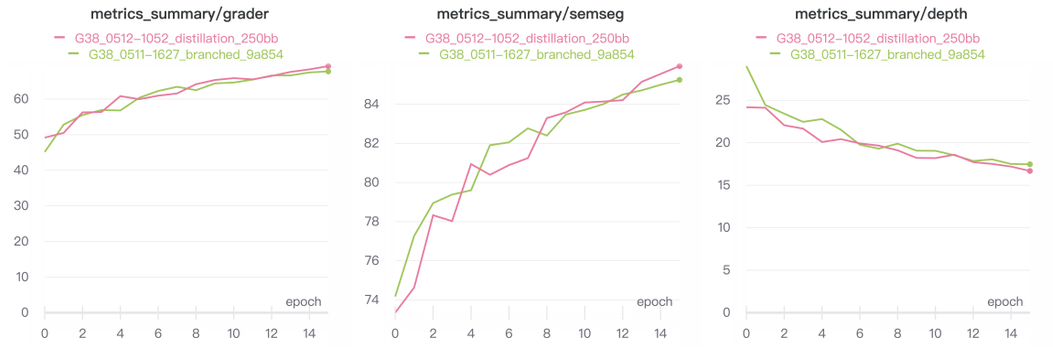
Table 6: Joint Architecture and Branched Architecture Summary

	Grader	Semseg	Depth	Run	Model Size	Duration
Joint Architecture	66.427	84.902	18.475	G38_0511–0853_v3p_93514	26716084	7h18m29s
Branched Architecture	67.778	85.246	17.468	G38_0511–1627_branched_9a854	32142356	9h1m37s

3 Task Distillation

Code for this architecture can be found in *mtl/models/model_distillation.py*. Add a command line with `--model_name 'distillation'` in *aws_train.sh* to use this model. With task distillation, the performance is improved from 67.778 to 69.268.

Similar to the branced architecture, we use different branches for different tasks. Furthermore, depth information and semantic information are related with each other. Based on this assumption, the self-attention part is designed for extracting useful information from other tasks. We also use additional decoders to restore information from previous information about this task and distillated information from other tasks. What's worse, the model size and computation cost will increase a lot bacause the model has two more self-attention parts and two more decoders parts for task distillation.



(a) (b) (c)

Figure 7: Comparision of w/o and w/ Task Distillation

Table 7: Branched Architecture w/o and w/ Task Distillation

	Grader	Semseg	Depth	Run	Model Size	Duration
w/o Task Distillation	67.778	85.246	17.468	G38_0511–1627_branched_9a854	32142356	9h1m37s
w Task Distillation	69.268	85.947	16.679	G38_0512–1052_distillation_250bb	36868136	14h34m2s

Finally, our results on the leaderboard are shown in Table 8:

Table 8: Final Results Summary

	Grader	Semseg	Depth	Run
Final Results	69.31 (22)	86.00 (21)	16.68 (20)	G38_0512–1052_distillation_adam_0.0001_250bb