Fast Neural Architecture Search of Compact Semantic Segmentation Models via Auxiliary Cells

Vladimir Nekrasov* Hao Chen* Chunhua Shen Ian Reid
The University of Adelaide, Australia
E-mail: {vladimir.nekrasov, hao.chen01, chunhua.shen, ian.reid}@adelaide.edu.au

Appendix A: Search Space

Decoder connectivity structure

Our fully-convolutional networks follow the encoder-decoder design paradigm. In particular, in place of the encoder we rely on an existing image classifier - here, MobileNet-v2 [8]. The decoder has access to 4 layers from the encoder with varying dimensions. To form connections inside the decoder part, we i.) first sample a pair of indices out of 4 possible choices with replacement, ii.) apply the same set of operations (*cell*) on each sample index, iii.) sum up the outputs (Fig. 1), and iv.) add the resultant layer into the sampling pool. In total, we repeat this process 3 times. Finally, all non-sampled summation outputs are concatenated, before being fed into a single 1×1 convolution to reduce the number of channels followed by the final classification layer.

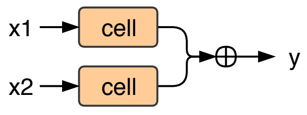


Figure 1 – Block structure of the decoder. The same cell operation is applied to two different layers specified by the connectivity configuration. If the two features have different size, the smaller one is scaled up via bilinear upsampling to match the larger one.

Cell structure

The cell structure is similarly generated via sampling a set of operations and corresponding indices. Nevertheless, there are several notable differences:

1. The operation at each position can vary;
2. A single operation is applied to the input without any aggregation operator;
3. After that, two indices and two operations are being sampled with replacement, with the corresponding outputs being summed up (this is repeated 3 times);
4. The outputs of each operation along with their summation layer are added into the sampling pool.

An example of the cell structure with its complete search space is illustrated in Fig. 2.

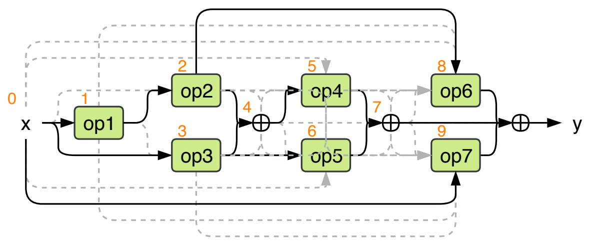


Figure 2 – Example cell structure of the decoder. The digit at the upper left corner of each operator is the index of the intermediate features. The cell is designed to utilize these features by skip connections. Except the first operator, other operators can be connected from any previous outputs. The solid black lines indicate the used paths and dashed grey lines are other unused possible paths. The cell configuration to generate the above cell is [op1, [1, 0, op2, op3], [4, 3, op4, op5], [2, 0, op6, op7]].

Architecture description

We use a list of integers to encode the architecture found by the controller, corresponding to the output sequence of the RNN. Specifically, the list describes the connectivity structure and the cell configuration. For example, the following connectivity structure $[[c_1, c_2], [c_3, c_4], [c_5, c_6]]$ contains three pairs of digits, indicating the input index c_k of a corresponding layer in the sampling pool. The cell configuration, $[o_1, [i_2, i_3, o_2, o_3], [i_4, i_5, o_4, o_5], [i_6, i_7, o_6, o_7]$, comprises the first operation o_1 followed by three cell branches with the operation o_j applied on the index i_j.

We provide the description of operations in Table 1, and visualise the discovered structures in Fig. 3 (*arch0*), Fig. 4 (*arch1*), and Fig. 5 (*arch2*). Note that inside the cell only the final summation operator is displayed as intermediate summations would lead to identical structures.

*Equal contribution.

Index	Abbreviation	Description
0	conv1x1	conv 1×1
1	conv3x3	conv 3×3
2	sep3x3	separable conv 3×3
3	sep5x5	separable conv 5×5
4	gap	global average pooling followed by upsampling and conv 1×1
5	conv3x3 rate 3	conv 3×3 with dilation rate 3
6	conv3x3 rate 12	conv 3×3 with dilation rate 12
7	sep3x3 rate 3	separable conv 3×3 with dilation rate 3
8	sep5x5 rate 6	separable conv 5×5 with dilation rate 6
9	skip	skip-connection
10	zero	zero-operation that effectively nullifies the path

Table 1 – Operation indices and abbreviations used to describe the cell configuration.

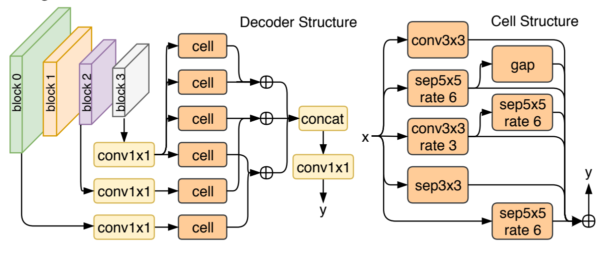


Figure 3 – arch0: [[[3, 3], [3, 2], [3, 0]], [8, [0, 0, 5, 2], [0, 2, 8, 8], [0, 5, 1, 4]]]

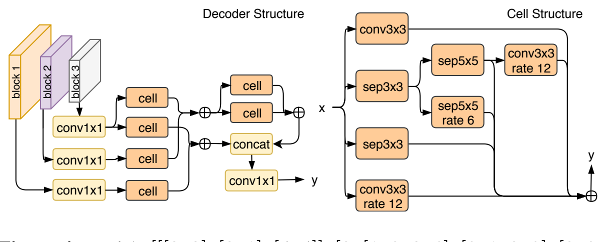


Figure 4 – arch1: [[[2, 3], [3, 1], [4, 4]], [2, [1, 0, 3, 6], [0, 1, 2, 8], [2, 0, 6, 1]]]

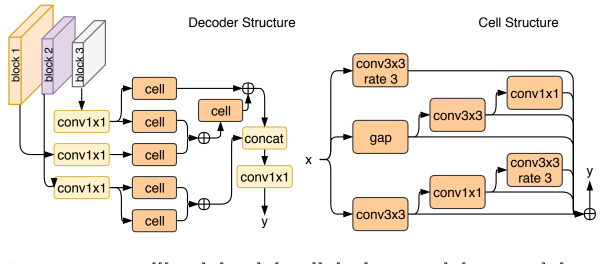


Figure 5 – arch2: [[[1, 3], [4, 3], [2, 2]], [5, [0, 0, 4, 1], [3, 2, 0, 1], [5, 6, 5, 0]]]

Appendix B: Experimental results

Semantic Segmentation

We start training with the learning rates of $1e$-3 and $3e$-3 - for the encoder and the decoder, respectively. The encoder weights are updated using SGD with the momentum value of 0.9, whereas for the decoder part we rely on Adam [3] with default parameters of β_1=0.9, β_2=0.99 and ϵ=0.001. We exploit the batch size of 64, evenly divided over two 1080Ti GPU cards. Each image in the batch is randomly scaled in the range of $[0.5, 2.0]$, randomly mirrored, before being randomly cropped and padded to the size of 450×450. During training, in order to calculate the loss term, we upsample the logits to the size of the target mask.

In addition to the results presented in the main text, we provide per-class intersection-over-union values across the models in Table 2.

Pose estimation

For pose estimation, we crop the human instance with fixed aspect ratios, 1:1 for MPII [1] and 3:4 for COCO [5]. Following Xiao *et al.* [9], the bounding box is further resized such that the longer side is equal to 256. For MPII, $\pm25\%$ scale, ±30 degree rotation and random flip are used for data augmentation. The scale and rotation factors for COCO are $\pm30\%$ and ±40 degrees, respectively. We generate keypoint heatmaps of output stride 4 with Gaussian distribution with $\sigma = 2$. The MobileNet-v2 encoder is initialised from ImageNet. We use the Adam optimiser with the base learning rate of $1e{-}3$, and reduce it by 10 after epochs 90 and 120. The training terminates at the epoch 140. We use the batch size of 128 evenly split between two 1080Ti GPU cards.

We provide detailed quantitative results on MPII in Table 3 and COCO in Table 4 along with a few qualitative examples on Fig. 6. The discovered architectures are able to infer correctly the location of the majority of keypoints (rows 1, 2, 4, 5) while failing on a more difficult input image along with other models (row 3).

Depth estimation

For depth estimation, we start training with the learning rates of $1e$-3 and $7e$-3 - for the encoder and the decoder, respectively. For both we use SGD with the momentum value of 0.9, and anneal the learning rates via the *'Poly'* schedule: $lr * (1 - \frac{epoch}{400})^{0.9}$. The training is stopped after 300 epochs. We exploit the batch size of 32, evenly divided over two 1080Ti GPU cards. Each image in the batch is randomly scaled in the range of $[0.5, 2.0]$, randomly mirrored, before being randomly cropped and padded to the size of 500×500. We upsample the logits to the size of the target mask and use the inverse Huber loss [4] for optimisation, ignoring pixels with missing depth measurements.

We visualise qualitative results on the validation set in Fig. 7.

Model	bg	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
DeepLab-v3 [8]	0.94	0.873	0.416	0.849	0.647	**0.753**	0.937	0.86	**0.904**	0.391	**0.893**	0.564	**0.847**	**0.892**	0.831	0.844	0.578	0.859	0.525	0.852	0.677	0.759
RefineNet-LW [7]	0.942	**0.895**	0.594	0.872	0.761	0.669	0.912	0.85	0.876	0.383	0.801	**0.605**	0.804	0.886	0.835	0.854	**0.603**	0.843	0.479	0.834	**0.703**	0.762
Ours (arch0)	**0.947**	0.885	0.558	0.885	0.748	0.74	**0.944**	0.868	0.898	**0.429**	0.863	0.604	0.846	0.842	0.866	0.86	0.592	**0.869**	**0.593**	**0.875**	0.669	**0.780**
Ours (arch1)	0.944	0.888	**0.615**	0.866	**0.781**	0.733	0.933	0.865	0.894	0.394	0.828	0.603	0.833	0.848	0.854	0.855	0.568	0.829	0.555	0.85	0.662	0.771
Ours (arch2)	**0.947**	0.873	0.589	**0.887**	0.753	0.75	0.943	**0.885**	0.895	0.372	0.829	0.635	0.845	0.832	**0.867**	**0.866**	0.555	0.843	0.537	0.851	0.671	0.773

Table 2 – Per-class intersection-over-union on the validation set of PASCAL VOC.

Model	Head	Shoulder	Elbow	Wrist	Hip	Knee	Ankle	Mean	Mean@0.1
DeepLab-v3+ [2]	96.180	94.735	86.859	81.037	87.312	81.281	76.121	86.609	31.735
ResNet-50 [9]	**96.351**	**95.329**	**88.989**	**83.176**	**88.420**	**83.960**	**79.594**	**88.532**	**33.911**
Ours (arch0)	95.873	94.378	86.296	80.195	87.139	81.160	75.885	86.526	31.435
Ours (arch1)	96.317	94.548	86.501	80.932	87.242	81.583	77.374	86.971	31.951
Ours (arch2)	96.146	94.769	87.097	80.574	87.848	81.382	77.586	87.119	31.782

Table 3 – Per-keypoint pose estimation results on the validation set of MPII.

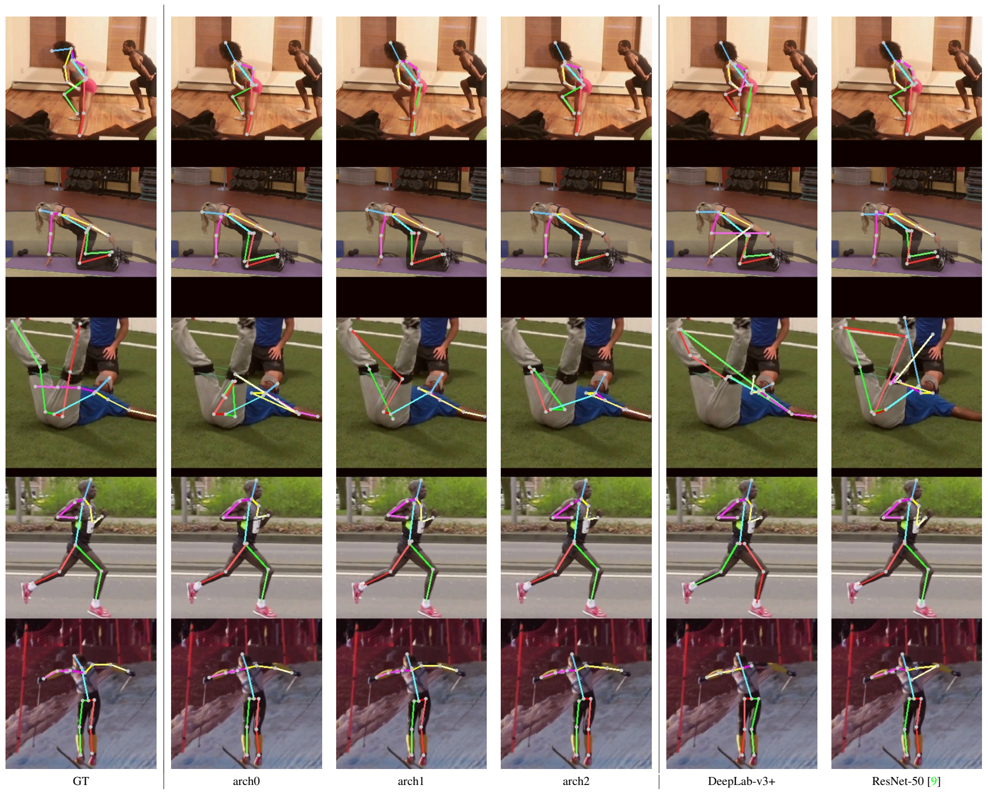


Figure 6 – Inference results of our models (*arch0*, *arch1*, *arch2*) on validation set of MPII, along with that of DeepLab-v3+-MobileNet-v2 and ResNet-50 [9].

Appendix C: JetsonTX2 runtime

During our experiments we observed a significant difference between models' runtime on JetsonTX2 and 1080Ti. To better understand it, we additionally measured runtime of each discovered architecture together with Light-Weight RefineNet [7] varying the input resolution.

As evident from Fig. 8, the models with a larger number of floating point operations (i.e., *Arch0* and *RF-LW*) do not scale well with the input resolution. The effect is even more pronounced on JetsonTX2, as been independently confirmed by an NVIDIA employer in a private conversation.

Model	AP	AP_{50}	AP_{75}	AP_m	AP_l	AR
DeepLab-v3+ [2]	0.668	**0.894**	0.740	0.641	0.707	0.700
ResNet-50 [9]	**0.704**	0.886	**0.783**	**0.671**	**0.772**	**0.763**
Ours (arch0)	0.658	**0.894**	0.730	0.631	0.701	0.691
Ours (arch1)	0.659	0.884	0.729	0.633	0.698	0.694
Ours (arch2)	0.659	0.890	0.729	0.631	0.700	0.693

Table 4 – Pose estimation results on the validation set of COCO2017. We report average precision (*AP)* and average recall (*AR*). AP_{50} and AP_{75} stand for average precision computed with the object keypoint similarity (OKS) values of 0.5 and 0.75, respectively, whereas AP_m and AP_l are average precision metrics as measured at medium and large area ranges.

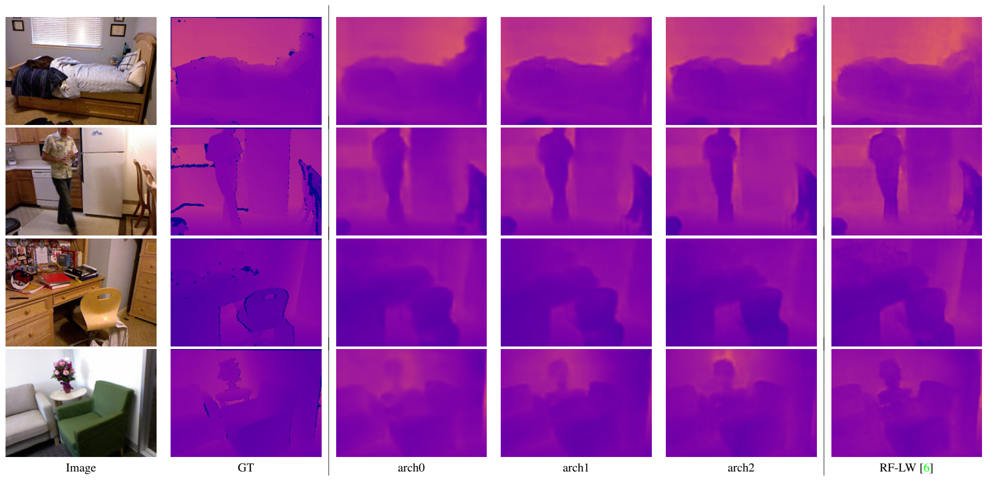
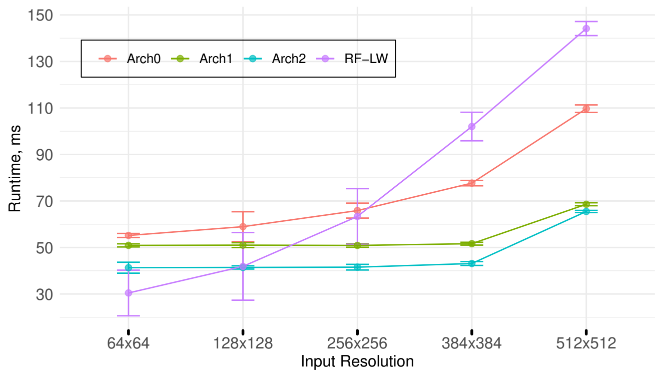
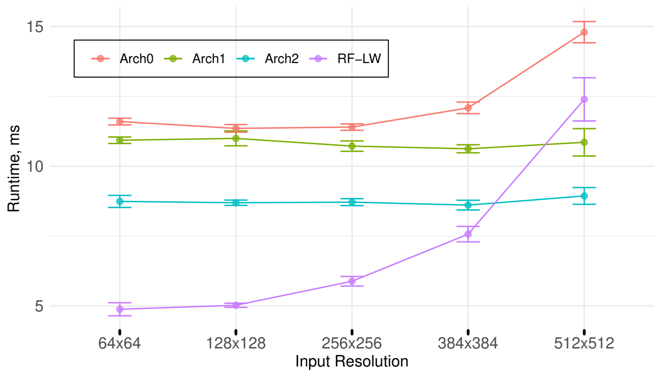


Figure 7 – Our depth estimation qualitative results on NYUDv2, along with that of Joint Light-Weight RefineNet [6]. Dark-blue pixels in ground truth are pixels with missing depth measurements.



(a) JetsonTX2



(b) 1080Ti

Figure 8 – Models' runtime on JetsonTX2 (**a**) and 1080Ti (**b**). We visualise mean together with standard deviation values over 100 passes of each model.

References

[1] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2014. 2

[2] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proc. Eur. Conf. Comp. Vis.*, 2018. 3, 4

[3] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv: Comp. Res. Repository*, abs/1412.6980, 2014. 2

[4] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. Deeper depth prediction with fully convolutional residual networks. In *Proc. Int. Conf. 3D Vision*, 2016. 2

[5] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. In *Proc. Eur. Conf. Comp. Vis.*, 2014. 2

[6] V. Nekrasov, T. Dharmasiri, A. Spek, T. Drummond, C. Shen, and I. D. Reid. Real-time joint semantic segmentation and depth estimation using asymmetric annotations. *arXiv: Comp. Res. Repository*, abs/1809.04766, 2018. 4

[7] V. Nekrasov, C. Shen, and I. D. Reid. Light-weight refinenet for real-time semantic segmentation. In *Proc. British Machine Vis. Conf.*, 2018. 3

[8] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018. 1, 3

[9] B. Xiao, H. Wu, and Y. Wei. Simple baselines for human pose estimation and tracking. In *Proc. Eur. Conf. Comp. Vis.*, 2018. 2, 3, 4