Fast and Efficient DNN Deployment via Deep Gaussian Transfer Learning Appendix

1. Pseudo-code

The pseudo-codes of our framework are shown in Algorithm 1 and Algorithm 2. Note that the pseudo-codes focus on the transfer learning and optimum searching, while the model preparations are ignored.

Algorithm 1 DNN Deployment Flow via Deep Gaussian Transfer Learning

---

Input: A DNN model with L tasks, with the configuration set $\{\mathcal{D}_1, \cdots, \mathcal{D}_L\}$, and type set $\{t_1, \cdots, t_L\}$; N types of pre-trained DGP $\{G_1, \cdots, G_N\}$; size of the random set r, size of the tuning set s, size of the final deployed configurations e.

Output: The optimal configurations $\mathcal{D}^* = \{d_1^*, \cdots, d_L^*\}$.

1: $\mathcal{D}^* \leftarrow \emptyset$;
2: **for** $l = 1 \to L$ **do**
3: t_l = type of layer l;
4: $G'_l = \mathbf{TL}(G_{t_l}, \mathcal{D}_l, r, s)$; ▷ Algorithm 2
5: Run simulated annealing with G'_l as the performance estimator, record the explored best e configurations X;
6: Deploy X on hardware, get the real performance Y;
7: d_l^* is the configuration with best performance in X;
8: $\mathcal{D}^* \leftarrow \mathcal{D}^* \cup d_l^*$;
9: **end for**
10: **return** The optimal configuration set $\mathcal{D}^*$;

Algorithm 2 Transfer Learning – $\mathbf{TL}(G, \mathcal{D}, r, s)$

Input: A pre-trained DGP model G, the configuration space $\mathcal{D}$, size of the random set r, and size of the tuning set s.

Output: The tuned DGP model G'.

1: Randomly sample a set $\mathcal{R}$ from $\mathcal{D}$, with $|\mathcal{R}| = r$;
2: Pass $\mathcal{R}$ to G, to get the predicted performance $\mathcal{P}$;
3: Sort $\mathcal{P}$ and select the top s configurations $\mathcal{T}_X$;
4: Deploy $\mathcal{T}_X$ on hardware, get the real performance $\mathcal{T}_Y$;
5: Tune G with $\{\mathcal{T}_X, \mathcal{T}_Y\}$, according to Formulation (6), the tuned model is G^t;
6: **return** The tuned DGP model G^t;

---

2. Experimental Settings

The feature encodings for the models are listed in Table 1. Further, tile_f, tile_y, and tile_x are tiled into four axes, corresponding to the number of blocks, the number of virtual threads, the number of threads, and the number of loops in each thread. tile_rc, tile_ry, and tile_rx are tiled into two axes, corresponding to the number of threads, and the number of loops in each thread. Note that for the depthwise convolutional layers in MobileNet-v1, there are no tile_rc, tile_ry, and tile_rx because of the special structures of the depthwise convolutions. In all, for the common convolutional layers, the length of the feature vector is 20, and for the depthwise convolutional layers, the length of the feature vector is 14.

Details of the DNN models and layers used in our experiments are listed in Table 2, Table 3, Table 4, and Table 5, where "# of Space" is the number of configurations in the design space, "Group Index" is the index of the group to which this layer belongs, c represents the size of input features, k represents the size of the kernel, s represents the step size of the stride, p represents the size of the padding.

Table 1: Features Encodings

Feature	Description
tile_f	# of output channels
tile_y	height of input features
tile_x	width of input features
tile_rc	# of input channels
tile_ry	height of kernels
tile_rx	width of kernels
auto_unroll_max_step	Maxiumal # of steps in the loop to be unrolled automatically
unroll_explicit	whether add unroll pragmas explicitly in the generated CUDA code

Our DGP model is implemented based on GPyTorch [1]. The radial basis function is the kernel function. In our DGP model, there are two hidden layers with the hidden dimensions 10 and 14, and the number of inducing points is 128. During training, the optimizer is Adam, the learning rate is 0.04, and the maximal training epoch is 3000. The training batch size is 256. The learning rate is adjusted progressively according to the reductions of RMSE (root-mean-square-error) of the predicted GFLOPS. During the transfer learning stage, 40000 configurations are randomly sampled from the configuration space, and then the top 300 configurations are used to tune the model, *i.e.*, $r = 40000$, $s = 300$ while calling Algorithm 2. The maximal tuning epoch is 2000, and the learning rate is 0.04. To compensate for the losses of using our DGP model instead of the real hardware, the stopping criterion of our method is no performance improvements in 600 iterations, or at most 40000 configurations have been explored by our method. Since our DGP model is used in replacement of the real hardware, the computations of the configurations are much faster. Finally, after the searching process, the configurations with the top 100 predicted performance values (*i.e.*, $e = 100$ in Algorithm 1) are compiled and run on GPU. The configuration with the highest on-board performance is chosen as the final optimal configuration. The settings of AutoTVM are the same as CHAMELEON [2].

3. Experimental Results

In Section 4.3, the randomly sampled configurations and the configurations selected via our pre-trained DGP are compared. More results are plotted in the following figures: Figure 1, Figure 2, Figure 3, and Figure 4. For each task, 300 configurations are sampled, and the data are in descending order. The results show the efficiencies of our DGP model while choosing tuning sets for new tasks. The random method samples lots of invalid configurations on the hardware while doing no favor to the tuning of the model. In comparison, our DGP can choose more useful configurations and the GFLOPS values are continuous in the value space. Besides, the maximal GFLOPS values sampled by DGP are higher which would help introduce more information for the new prediction tasks.

GFLOPS results of the DNN models are plotted in Figure 5. The results show that our method wins on most layers.

References

[1] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson, "GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration," in *Proc. NIPS*, 2018. 1

[2] B. H. Ahn, P. Pilligundla, A. Yazdanbakhsh, and H. Esmaeilzadeh, "CHAMELEON: Adaptive code optimization for expedited deep neural network compilation," in *Proc. ICLR*, 2020. 2

Table 2: Details of AlexNet

Task Index	Name	# of Space	Group Index	Layer Description
1	`conv5`	570240	1	$c = 256 \times 13 \times 13$, $k = 256 \times 256 \times 13 \times 13$, $s = 1$, $p = 1$
2	`conv4`	1013760	1	$c = 384 \times 13 \times 13$, $k = 256 \times 384 \times 3 \times 3$, $s = 1$, $p = 1$
3	`conv3`	2580480	1	$c = 192 \times 13 \times 13$, $k = 384 \times 192 \times 3 \times 3$, $s = 1$, $p = 1$
4	`conv2`	22579200	2	$c = 64 \times 27 \times 27$, $k = 192 \times 64 \times 5 \times 5$, $s = 1$, $p = 2$
5	`conv1`	1032192	3	$c = 3 \times 224 \times 224$, $k = 64 \times 3 \times 11 \times 11$, $s = 4$, $p = 2$

Table 3: Details of ResNet-18

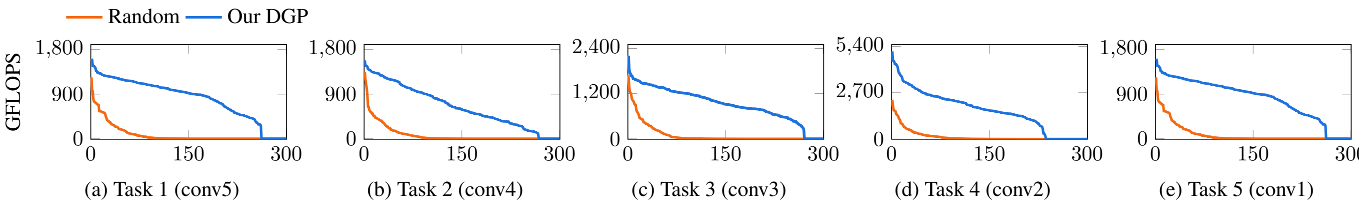
Task Index	Name	# of Space	Group Index	Layer Description
1	`rb4-conv2`	844800	1	$c = 512 \times 7 \times 7$, $k = 512 \times 512 \times 3 \times 3$, $s = 1$, $p = 1$
2	`rb4-conv1`	760320	2	$c = 256 \times 14 \times 14$, $k = 512 \times 256 \times 3 \times 3$, $s = 2$, $p = 1$
3	`rb3-conv2`	9123840	1	$c = 256 \times 14 \times 14$, $k = 256 \times 256 \times 3 \times 3$, $s = 1$, $p = 1$
4	`rb3-conv1`	8110080	2	$c = 128 \times 28 \times 28$, $k = 256 \times 128 \times 3 \times 3$, $s = 2$, $p = 1$
5	`rb2-conv2`	36864000	1	$c = 128 \times 28 \times 28$, $k = 128 \times 128 \times 3 \times 3$, $s = 1$, $p = 1$
6	`rb2-conv1`	32256000	2	$c = 64 \times 56 \times 56$, $k = 128 \times 64 \times 3 \times 3$, $s = 2$, $p = 1$
7	`rb1-conv1`	90316800	1	$c = 64 \times 56 \times 56$, $k = 64 \times 64 \times 3 \times 3$, $s = 1$, $p = 1$
8	`conv0`	79027200	3	$c = 3 \times 224 \times 224$, $k = 64 \times 3 \times 7 \times 7$, $s = 2$, $p = 3$
9	`rb1-sc`	22579200	4	$c = 64 \times 56 \times 56$, $k = 64 \times 64 \times 1 \times 1$, $s = 1$, $p = 1$
10	`rb2-sc`	8064000	4	$c = 64 \times 56 \times 56$, $k = 128 \times 64 \times 1 \times 1$, $s = 1$, $p = 1$
11	`rb3-sc`	2027520	4	$c = 128 \times 28 \times 28$, $k = 256 \times 128 \times 1 \times 1$, $s = 1$, $p = 1$
12	`rb4-sc`	190080	4	$c = 256 \times 14 \times 14$, $k = 512 \times 256 \times 1 \times 1$, $s = 1$, $p = 1$

Table 4: Details of VGG-16

Task Index	Name	# of Space	Group Index	Layer Description
1	`conv4-3`	13516800	1	$c = 512 \times 14 \times 14$, $k = 512 \times 512 \times 3 \times 3$, $s = 1$, $p = 1$
2	`conv4-2`	8448000	1	$c = 512 \times 28 \times 28$, $k = 512 \times 512 \times 3 \times 3$, $s = 1$, $p = 1$
3	`conv4-1`	76032000	1	$c = 256 \times 28 \times 28$, $k = 512 \times 256 \times 3 \times 3$, $s = 1$, $p = 1$
4	`conv3-2`	228096000	1	$c = 256 \times 56 \times 56$, $k = 256 \times 256 \times 3 \times 3$, $s = 1$, $p = 1$
5	`conv3-1`	202752000	1	$c = 128 \times 56 \times 56$, $k = 256 \times 128 \times 3 \times 3$, $s = 1$, $p = 1$
6	`conv2-2`	451584000	1	$c = 128 \times 112 \times 112$, $k = 128 \times 128 \times 3 \times 3$, $s = 1$, $p = 1$
7	`conv2-1`	395136000	1	$c = 64 \times 112 \times 112$, $k = 128 \times 64 \times 3 \times 3$, $s = 1$, $p = 1$
8	`conv1-2`	708083712	1	$c = 64 \times 224 \times 224$, $k = 64 \times 64 \times 3 \times 3$, $s = 1$, $p = 1$
9	`conv1-1`	202309632	1	$c = 3 \times 224 \times 224$, $k = 64 \times 3 \times 3 \times 3$, $s = 1$, $p = 1$

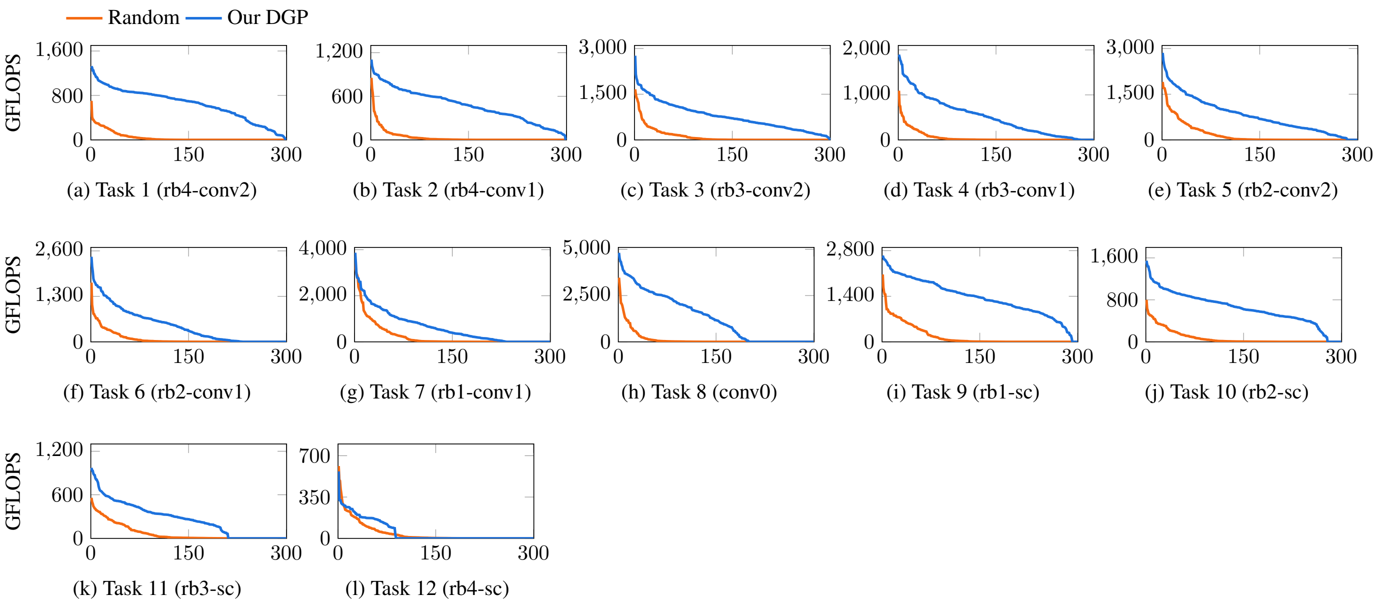
Table 5: Details of MobileNet-v1

Task Index	Name	# of Space	Group Index	Layer Description
1	`sp-13-conv2`	302016	1	$c = 1024 \times 7 \times 7$, $k = 1024 \times 1024 \times 1 \times 1$, $s = 1$, $p = 0$
2	`sp-13-dp1`	27456	2	$c = 1024 \times 7 \times 7$, $k = 1024 \times 1 \times 3 \times 3$, $s = 2$, $p = 1$
3	`sp-12-conv2`	274560	1	$c = 512 \times 7 \times 7$, $k = 1024 \times 512 \times 1 \times 1$, $s = 1$, $p = 0$
4	`sp-12-dp1`	21120	2	$c = 512 \times 14 \times 14$, $k = 512 \times 1 \times 3 \times 3$, $s = 2$, $p = 1$
5	`sp-7-conv2`	3379200	1	$c = 512 \times 14 \times 14$, $k = 512 \times 512 \times 1 \times 1$, $s = 4$, $p = 0$
6	`sp-7-dp1`	337920	3	$c = 512 \times 14 \times 14$, $k = 512 \times 1 \times 3 \times 3$, $s = 1$, $p = 1$
7	`sp-6-conv2`	3041280	1	$c = 256 \times 14 \times 14$, $k = 512 \times 256 \times 1 \times 1$, $s = 1$, $p = 0$
8	`sp-6-dp1`	253440	2	$c = 256 \times 28 \times 28$, $k = 256 \times 1 \times 3 \times 3$, $s = 2$, $p = 1$
9	`sp-5-conv2`	14256000	1	$c = 256 \times 28 \times 28$, $k = 256 \times 256 \times 1 \times 1$, $s = 1$, $p = 0$
10	`sp-5-dp1`	1584000	3	$c = 256 \times 28 \times 28$, $k = 256 \times 1 \times 3 \times 3$, $s = 1$, $p = 1$
11	`sp-4-conv2`	12672000	1	$c = 128 \times 28 \times 28$, $k = 256 \times 128 \times 1 \times 1$, $s = 1$, $p = 0$
12	`sp-4-dp1`	1152000	2	$c = 128 \times 56 \times 56$, $k = 128 \times 1 \times 3 \times 3$, $s = 2$, $p = 1$
13	`sp-3-conv2`	36864000	1	$c = 128 \times 56 \times 56$, $k = 128 \times 128 \times 1 \times 1$, $s = 1$, $p = 0$
14	`sp-3-dp1`	4608000	3	$c = 128 \times 56 \times 56$, $k = 128 \times 1 \times 3 \times 3$, $s = 1$, $p = 1$
15	`sp-2-conv2`	32256000	1	$c = 64 \times 56 \times 56$, $k = 128 \times 64 \times 1 \times 1$, $s = 1$, $p = 0$
16	`sp-2-dp-1`	3225600	2	$c = 64 \times 112 \times 112$, $k = 64 \times 1 \times 3 \times 3$, $s = 2$, $p = 1$
17	`sp-1-conv1`	59270400	1	$c = 32 \times 112 \times 112$, $k = 64 \times 32 \times 1 \times 1$, $s = 1$, $p = 0$
18	`sp-1-dp-1`	6585600	3	$c = 32 \times 112 \times 112$, $k = 32 \times 1 \times 3 \times 3$, $s = 1$, $p = 1$
19	`conv1`	52684800	4	$c = 3 \times 224 \times 224$, $k = 32 \times 3 \times 3 \times 3$, $s = 2$, $p = 1$



(a) Task 1 (conv5) (b) Task 2 (conv4) (c) Task 3 (conv3) (d) Task 4 (conv2) (e) Task 5 (conv1)

Figure 1: The tuning sets of AlexNet. The data are in descending order.

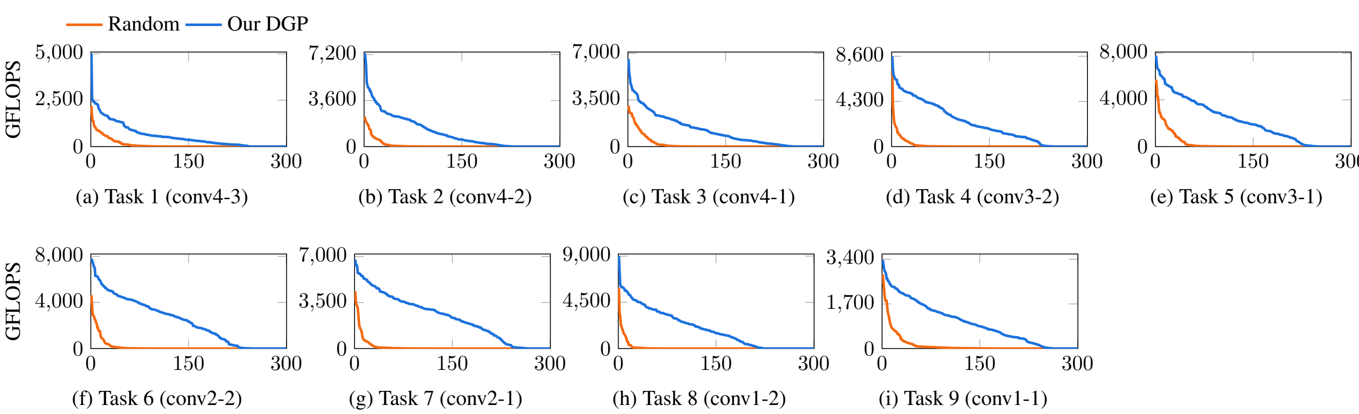


(a) Task 1 (rb4-conv2) (b) Task 2 (rb4-conv1) (c) Task 3 (rb3-conv2) (d) Task 4 (rb3-conv1) (e) Task 5 (rb2-conv2)

(f) Task 6 (rb2-conv1) (g) Task 7 (rb1-conv1) (h) Task 8 (conv0) (i) Task 9 (rb1-sc) (j) Task 10 (rb2-sc)

(k) Task 11 (rb3-sc) (l) Task 12 (rb4-sc)

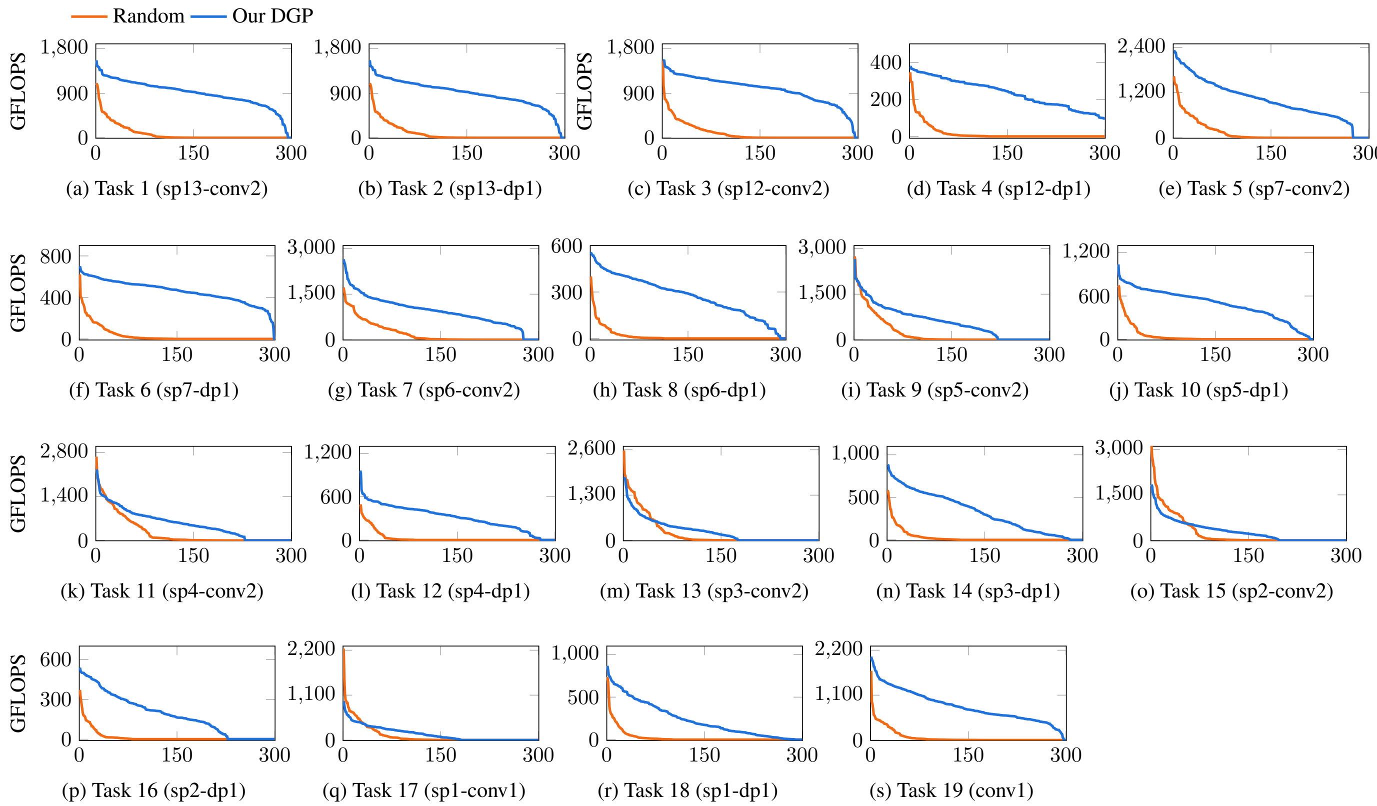
Figure 2: The tuning sets of ResNet-18. The data are in descending order.



(a) Task 1 (conv4-3) (b) Task 2 (conv4-2) (c) Task 3 (conv4-1) (d) Task 4 (conv3-2) (e) Task 5 (conv3-1)

(f) Task 6 (conv2-2) (g) Task 7 (conv2-1) (h) Task 8 (conv1-2) (i) Task 9 (conv1-1)

Figure 3: The tuning sets of VGG-16. The data are in descending order.



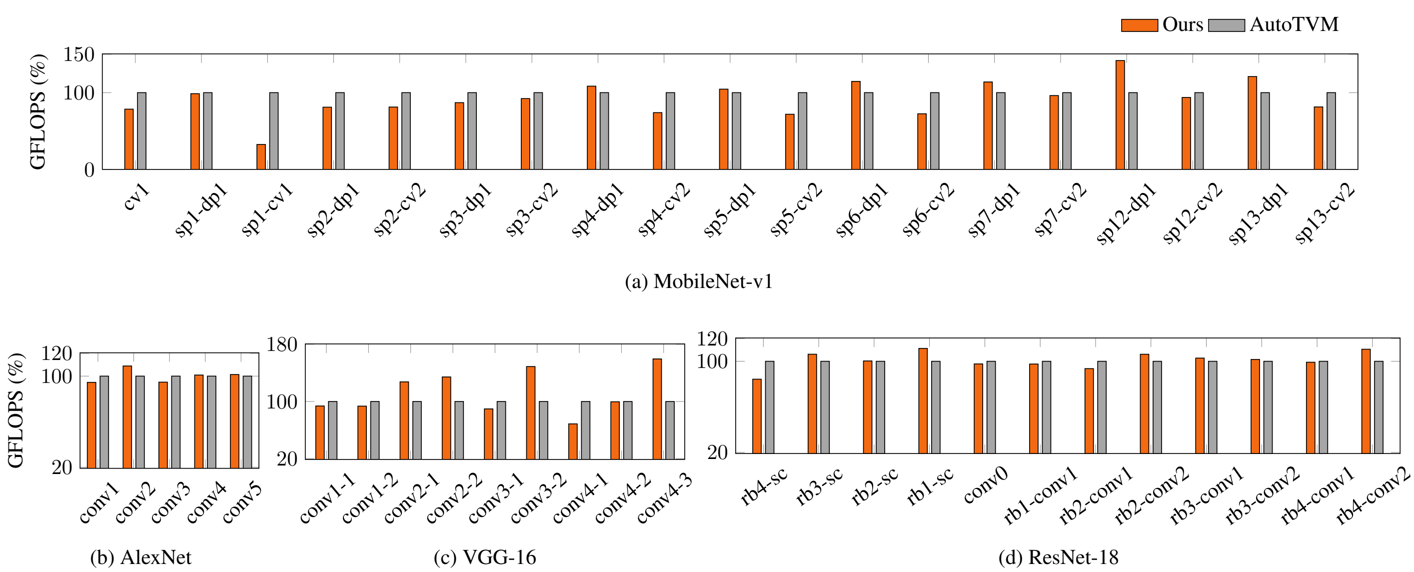
(a) Task 1 (sp13-conv2) (b) Task 2 (sp13-dp1) (c) Task 3 (sp12-conv2) (d) Task 4 (sp12-dp1) (e) Task 5 (sp7-conv2)

(f) Task 6 (sp7-dp1) (g) Task 7 (sp6-conv2) (h) Task 8 (sp6-dp1) (i) Task 9 (sp5-conv2) (j) Task 10 (sp5-dp1)

(k) Task 11 (sp4-conv2) (l) Task 12 (sp4-dp1) (m) Task 13 (sp3-conv2) (n) Task 14 (sp3-dp1) (o) Task 15 (sp2-conv2)

(p) Task 16 (sp2-dp1) (q) Task 17 (sp1-conv1) (r) Task 18 (sp1-dp1) (s) Task 19 (conv1)

Figure 4: The tuning sets of MobileNet. The data are in descending order.



(a) MobileNet-v1

(b) AlexNet (c) VGG-16 (d) ResNet-18

Figure 5: The ratios of the GFLOPS values of various DNN models.