Appendix:

The appendix is organized as follows:

– In Sec. A1, we provide additional details and comparisons to learned and non-learned TTA in other setups, *e.g.*, additional TTA policy or architectures.
– In Sec. A2, we provide additional image classification results of our approach on CIFAR10 and CIFAR100 datasets [24]. We also provide more detailed results on ImageNet for image classification, Cityscapes, and ADE20k for semantic segmentation.
– In Sec. A3, we provide additional implementation details of our method, including how to implement subsampling layers for each of the corresponding backbones.

A1 Additional details and comparison to TTA

A1.1 Details for integrating Ours with existing TTA methods.

Existing TTA methods take an input image and form an augmented image pool $\{\boldsymbol{I}^{(a)}_{\texttt{aug}}\}_{a=1}^{B_{\texttt{tta}}}$ to make a prediction by

$$\hat{\boldsymbol{y}} = T\left(\left\{\hat{\boldsymbol{y}}^{(a)}\right\}_{a=1}^{B_{\texttt{tta}}}\right), \text{where } \hat{\boldsymbol{y}}^{(a)} = C_\phi(F_\theta(\boldsymbol{I}^{(a)}_{\texttt{aug}}; \boldsymbol{0})), \tag{A13}$$

where T is an aggregation function (different from ours) merging all logits resulting from $B_{\texttt{tta}}$ augmented images with default $\boldsymbol{s} = \boldsymbol{0}$. To integrate Ours with existing TTA methods, we perform our search and aggregation method *for each* augmented image to make the prediction $\hat{\boldsymbol{y}}^{(a)}$ in Eq. (A13), *i.e.*,

$$\hat{\boldsymbol{y}}^{(a)} = C_\phi\left(A(\mathcal{F}^{(a)})\right), \text{ where } \mathcal{F}^{(a)} = \{F(\boldsymbol{I}^{(a)}_{\texttt{aug}}; \boldsymbol{s}) \mid \boldsymbol{s} \in \hat{\mathcal{S}}\} \tag{A14}$$

following a simple aggregation A as described in the main paper. We use this described approach in our experiments when incorporating our approach with existing TTA methods.

A1.2 Learned TTA methods

A different TTA policy. We conduct additional experiments following another TTA policy setting proposed by Shanmugam et al. [52], *i.e.*, the `standard` TTA policy. The `standard` TTA consists of the following data transformations: `Flip`, `Scale`, and `FiveCrop`. The original `standard` policy fixes $B_{\texttt{tta}}$ at 30. We increase the possible range of $B_{\texttt{tta}}$ for `standard` policy to cover up to 190. Please refer to Sec. A3.2 for the details.

We report the comparison between the baseline TTA methods with and without our learned approach in Tab. A1 and Tab. A2 on ImageNet and Flowers102 respectively. We observe that, unlike our approach, the performance of TTA baselines is highly dependent on the TTA policy. Although the `standard` policy

Table A1: Comparison to TTA methods on ImageNet with standard [52] TTA policies. We evaluate on ImageNet under $B_{\texttt{total}} \in \{30, 100, 150\}$ with various model architectures. For each $B_{\texttt{total}}$, we report the top-1 Acc. of baseline TTA methods with and without our learned approach. Whichever is the better one is bolded. The results of our learned procedure are highlighted.

TTA	Ours	ResNet18			ResNet50			MobileNetV2			InceptionV3		
		30	100	150	30	100	150	30	100	150	30	100	150
GPS [38]	✗	70.38	70.38	70.38	76.37	76.37	76.37	72.27	72.27	72.27	71.74	71.74	71.74
	✓	**70.60**	**70.72**	**70.69**	**76.67**	**76.81**	**76.84**	**72.38**	**72.64**	**72.59**	**72.13**	**72.29**	**72.30**
ClassTTA [52]	✗	70.06	67.77	65.46	76.26	74.55	72.65	71.79	69.19	66.74	71.77	70.54	70.07
	✓	**70.70**	**70.73**	**70.69**	**76.76**	**76.83**	**76.85**	**72.32**	**72.34**	**72.29**	**72.26**	**72.43**	**72.41**
AugTTA [52]	✗	70.78	70.83	70.82	76.68	76.67	76.66	72.59	72.59	**72.61**	72.22	**72.99**	**73.02**
	✓	**70.89**	**70.85**	**70.86**	**76.74**	**76.82**	**76.86**	**72.60**	**72.69**	72.58	**72.42**	72.51	72.46

Table A2: Comparison to TTA methods on Flowers102 with standard [52] TTA policies. We evaluate on Flowers102 under $B_{\texttt{total}} \in \{30, 100, 150\}$ with various model architectures. For each $B_{\texttt{total}}$, we report the top-1 Acc. of baseline TTA methods with and without our learned approach. Whichever is the better one is bolded. The results of our learned procedure are highlighted.

TTA	Ours	ResNet18			ResNet50			MobileNetV2			InceptionV3		
		30	100	150	30	100	150	30	100	150	30	100	150
GPS [38]	✗	**89.07**	89.07	89.07	91.07	91.07	91.07	89.80	89.80	89.80	87.64	87.64	87.64
	✓	89.01	**89.22**	**89.19**	**91.17**	**91.28**	**91.28**	**89.90**	**90.06**	**90.11**	**87.66**	**87.71**	**87.75**
ClassTTA [52]	✗	88.62	87.40	83.28	90.78	89.09	86.29	89.30	87.49	84.35	**87.77**	86.63	83.75
	✓	**89.07**	**89.22**	**89.15**	**91.19**	**91.28**	**91.36**	**89.77**	**89.89**	**90.00**	87.53	**87.66**	**87.53**
AugTTA [52]	✗	88.88	88.97	87.40	**91.20**	90.80	89.41	**90.05**	89.93	88.19	**87.74**	87.10	86.91
	✓	**89.10**	**89.23**	**89.15**	91.13	**91.28**	**91.28**	89.92	**90.10**	**90.11**	87.61	**87.75**	**87.79**

improves the performance of TTA baselines, our approach shows a consistent gain over them, except for InceptionV3 with AugTTA. As shown in Tab. A1, on average, we improve the top-1 Acc. of TTA baselines by 0.9% on ImageNet. Our approach improves GPS on average by 0.4%, ClassTTA by 2.5%, and AugTTA by 0.1%. As shown in Tab. A2, on average, we improve the top-1 Acc. of TTA baselines by 0.6% on Flowers102. Specifically, Our approach improves GPS on average by 0.1%, ClassTTA by 1.3%, and AugTTA by 0.2%.

More architectures. In Tab. A3, we report the comparison of our learned approach to baseline TTA methods on more architectures, including ResNext50 [67], ShuffleNetV2 [39], Swin [34], and SwinV2 [33]. Specifically, we use these variants from torchvision [45]: `resnext50_32x4d` for ResNext50, `shufflenet_v2_x1_0` for ShuffleNetV2, `swin_t` for Swin, and `swin_v2_t` for SwinV2. We observe consistent gain across all the models on ImageNet.

Computation budget. We compare the computation budget (MACs, and latency) on an image of resolution 224px by 224px between ours and baseline methods. We measure the latency (ms/img) from end-to-end over 10 runs on an Nvidia A6000 GPU. We report the results of ResNet18 in Tab. A4 and MobileNetV2 in Tab. A5. While GPS [38] has less budget and latency, it does not achieve the best performance and does not scale with a higher budget as shown

Table A3: Comparison to TTA methods on ImageNet with expanded [52] TTA policies. We evaluate on ImageNet under $B_{\texttt{total}} \in \{30, 100, 150\}$ with various model architectures. For each $B_{\texttt{total}}$, we report the top-1 Acc. of baseline TTA methods with and without our learned approach. Whichever is the better one is bolded. The results of our learned procedure are highlighted.

TTA	Ours	ResNext50			ShuffleNetV2			Swin			SwinV2		
		30	100	150	30	100	150	30	100	150	30	100	150
GPS [38]	✗	78.01	78.01	78.01	70.17	70.17	70.17	81.35	81.35	81.35	81.31	81.31	81.31
	✓	**78.18**	**78.32**	**78.30**	**70.44**	**70.35**	**70.14**	**81.42**	**81.41**	**81.42**	**81.35**	**81.44**	**81.46**
ClassTTA [52]	✗	77.03	76.70	75.45	69.01	69.02	68.57	80.63	80.65	80.80	80.56	80.54	80.57
	✓	**78.10**	**78.15**	**78.20**	**70.36**	**70.15**	**70.07**	**81.52**	**81.50**	**81.54**	**81.39**	**81.39**	**81.38**
AugTTA [52]	✗	78.09	78.13	78.08	70.37	**70.49**	**70.40**	81.42	81.50	81.38	81.28	81.41	81.25
	✓	**78.14**	**78.23**	**78.28**	**70.51**	70.35	70.25	**81.49**	**81.51**	**81.52**	**81.40**	**81.42**	**81.42**

in the previous experiments. In terms of MACs, our approach requires more operations than ClassTTA [52] and AugTTA [52] on a lower budget. However, since our approach operates on downsampling layers only and shares most computation on non-downsampling layers, the overall latency is more or less similar to theirs.

A1.3 Non-learned TTA methods

For non-learned TTA methods, we follow Shanmugam et al. [52] and compared the `MeanTTA` and `MaxTTA` aggregation methods. `MeanTTA` computes the average over all the augmented logits, while `MaxTTA` picks the highest logit value for each class over all the augmented logits. We report the comparison of our non-learned approach to non-learned TTA methods, *i.e.*, we use Eq. (7) and Eq. (8) for `Aggregation` and Eq. (12) for `Criterion`. As shown in Tab. A6, we do not find it beneficial to use non-learned TTA methods with the expanded policy. We directly compare our non-learned approach without integrating TTA, *i.e.*, $B_{\texttt{tta}} = 1$, to non-learned TTA methods. Our non-learned approach outperforms `MeanTTA` and `MaxTTA` without needing to increase the computation budget.

In Tab. A7, we report the non-learned TTA methods with or without our non-learned approach using the standard policy. We observe that although `Mean` and `MaxTTA` work on a smaller budget, they do not scale with higher budgets. Our non-learned approach outperforms `MeanTTA` under most budget settings while `MaxTTA` is an ineffective non-learned TTA method.

A1.4 Budget configuration

With the integration approach described in Sec. A1.1, the total budget $B_{\texttt{total}} = B_{\texttt{tta}}\ times B_{\texttt{ours}}$ is a product of the budget for existing TTA methods $B_{\texttt{tta}}$ and ours $B_{\texttt{ours}}$. We have the flexibility of balancing between the two to achieve the same total budget. Here, we document the choices used in our experiments.

– For Tab. 1, Tab. 2, Tab. A1, Tab. A2, Tab. A3, Tab. A4, and Tab. A5, we use the following settings. When $B_{\texttt{total}} = 30$, we use $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (10, 3)$

Table A4: Comparison of computation budget of ResNet18 on ImageNet. We report the comparison of top-1 Acc. , MACs (G), and the latency (ms/img) between baseline TTA methods w/ and w/o our learned approach.

$B_{\texttt{total}}$	TTA	Ours	Acc ↑	MACs ↓		Latency ↓	
1	✗	✗	69.76	1.8	(1.0×)	1.6	(1.0×)
30	GPS	✗	70.51	**5.5**	**(3.0×)**	**4.2**	**(2.7×)**
		✓	**70.74**	20.8	(11.4×)	8.9	(5.7×)
	ClassTTA	✗	69.09	**54.7**	**(30.0×)**	43.3	(27.9×)
		✓	**70.37**	69.3	(38.0×)	**25.2**	**(16.2×)**
	AugTTA	✗	70.55	**54.7**	**(30.0×)**	54.0	(34.8×)
		✓	**70.75**	69.3	(38.0×)	**31.7**	**(20.4×)**
100	GPS	✗	70.51	**5.5**	**(3.0×)**	**4.0**	**(2.6×)**
		✓	**70.74**	50.3	(27.6×)	24.3	(15.6×)
	ClassTTA	✗	68.23	182.4	(100.0×)	155.8	(100.3×)
		✓	**70.36**	**167.7**	**(91.9×)**	**118.8**	**(76.5×)**
	AugTTA	✗	70.66	182.4	(100.0×)	141.4	(91.0×)
		✓	**70.79**	**167.7**	**(91.9×)**	**86.2**	**(55.5×)**
150	GPS	✗	70.51	**5.5**	**(3.0×)**	**3.9**	**(2.5×)**
		✓	**70.69**	76.1	(41.7×)	39.7	(25.6×)
	ClassTTA	✗	66.40	273.6	(150.0×)	236.4	(152.1×)
		✓	**70.37**	**253.6**	**(139.0×)**	**159.1**	**(102.4×)**
	AugTTA	✗	70.28	273.6	(150.0×)	244.0	(157.0×)
		✓	**70.74**	**253.6**	**(139.0×)**	**131.6**	**(84.7×)**

with ours and $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (30, 1)$ without ours. When $B_{\texttt{total}} = 100$, we use $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (10, 10)$ with ours and $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (100, 1)$ without ours. When $B_{\texttt{total}} = 150$, we use $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (10, 15)$ with ours and $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (150, 1)$ without ours.

– For Tab. A6, we use the following settings. When $B_{\texttt{total}} = 30$, we use $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (1, 30)$ with ours and $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (30, 1)$ without ours. When $B_{\texttt{total}} = 75$, we only use $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (75, 1)$ without ours. When $B_{\texttt{total}} = 150$, we only use $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (150, 1)$ without ours.

– For Tab. A7, we use the following settings. When $B_{\texttt{total}} = 30$, we use $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (15, 2)$ with ours and $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (30, 1)$ without ours. When $B_{\texttt{total}} = 75$, we use $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (15, 5)$ with ours and $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (75, 1)$ without ours. When $B_{\texttt{total}} = 150$, we use $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (15, 10)$ with ours and $(B_{\texttt{tta}}, B_{\texttt{ours}}) = (150, 1)$ without ours.

Table A5: Comparison of computation budget of MobileNetV2 on ImageNet. We report the comparison of top-1 Acc. , MACs (G), and the latency (ms/img) between baseline TTA methods w/ and w/o our learned approach.

$B_{\texttt{total}}$	TTA	Ours	Acc ↑	MACs ↓		Latency ↓	
1	✗	✗	71.88	0.3	(1.0×)	2.4	(1.0×)
30	GPS	✗	72.24	**1.0**	**(3.0×)**	**7.6**	**(3.1×)**
		✓	**72.37**	3.5	(10.7×)	11.8	(4.9×)
	ClassTTA	✗	70.58	**9.8**	**(30.0×)**	87.0	(36.0×)
		✓	**71.44**	11.6	(35.6×)	**40.9**	**(16.9×)**
	AugTTA	✗	72.33	**9.8**	**(30.0×)**	86.2	(35.7×)
		✓	**72.41**	11.7	(35.6×)	**46.0**	**(19.1×)**
100	GPS	✗	72.24	**1.0**	**(3.0×)**	**7.4**	**(3.1×)**
		✓	**72.61**	9.2	(28.1×)	38.9	(16.1×)
	ClassTTA	✗	69.97	32.8	(100.0×)	260.6	(108.0×)
		✓	**71.68**	**30.7**	**(93.6×)**	**140.4**	**(58.2×)**
	AugTTA	✗	72.42	32.8	(100.0×)	270.9	(112.3×)
		✓	**72.62**	**30.7**	**(93.6×)**	**120.9**	**(50.1×)**
150	GPS	✗	72.24	**1.0**	**(3.0×)**	**8.5**	**(3.5×)**
		✓	**72.58**	14.6	(44.5×)	69.5	(28.8×)
	ClassTTA	✗	67.81	49.1	(150.0×)	481.8	(199.7×)
		✓	**71.63**	**48.6**	**(148.4×)**	**235.5**	**(97.6×)**
	AugTTA	✗	72.46	49.1	(150.0×)	435.9	(180.6×)
		✓	**72.58**	**48.6**	**(148.4×)**	**246.9**	**(102.3×)**

Table A6: Comparison to non-learned TTA methods on ImageNet with expanded [52] TTA policies. We evaluate on ImageNet under $B_{\texttt{total}} \in \{30, 100, 150\}$ with various model architectures. For each $B_{\texttt{total}}$, we report the top-1 Acc. of non-learned TTA methods with and without our non-learned approach. Whichever is the better one is bolded. The results of our non-learned procedure are highlighted.

Method	ResNet18			ResNet50			MobileNetV2			InceptionV3		
	30	75	150	30	75	150	30	75	150	30	75	150
MeanTTA	67.23	68.37	67.81	73.88	74.77	74.30	68.72	69.98	69.44	70.46	70.71	70.51
MaxTTA	66.31	66.07	65.60	71.79	71.52	71.12	68.20	67.92	67.32	64.95	65.21	64.98
Ours	**70.27**	–	–	**76.54**	–	–	**72.35**	–	–	**71.45**	–	–

A2 Additional results

A2.1 CIFAR10 & CIFAR100

We use the publicly available implementation from PyTorch CIFAR Models (https://github.com/chenyaofo/pytorch-cifar-models), which supports a variety of pre-trained models on CIFAR10 and CIFAR100. Specifically, there are

Table A7: Comparison to non-learned TTA methods on ImageNet with standard [52] TTA policies. We evaluate on ImageNet under $B_{\texttt{total}} \in \{30, 100, 150\}$ with various model architectures. For each $B_{\texttt{total}}$, we report the top-1 Acc. of non-learned TTA methods with and without our non-learned approach. Whichever is the better one is bolded. The results of our non-learned procedure are highlighted.

TTA	Ours	ResNet18			ResNet50			MobileNetV2			InceptionV3		
		30	75	150	30	75	150	30	75	150	30	75	150
MeanTTA	✗	**70.90**	70.38	68.06	76.65	75.99	73.57	**72.65**	72.00	68.98	**71.99**	**71.61**	70.44
	✓	70.70	**70.84**	**70.81**	**76.71**	**76.76**	**76.83**	72.61	**72.57**	**72.71**	71.44	71.58	**71.98**
MaxTTA	✗	70.02	69.70	68.39	76.24	75.62	72.86	72.13	71.79	70.10	**71.42**	69.93	67.27
	✓	**70.24**	**70.45**	**70.46**	**76.49**	**76.51**	**76.66**	**72.33**	**72.43**	**72.54**	71.10	**71.33**	**71.76**

19 models in total, including ResNet [18], VGG [54], RepVGG [12], MobileNet [50], and ShuffleNet [75] architectures. We report the experiments on all of them.

As in the main paper, we report the top-1 classification accuracy vs. the budget $B_{\texttt{ours}}$ in Fig. A1 and Fig. A2 for CIFAR10 and CIFAR100 respectively. On both CIFAR10 and CIFAR100 datasets, we observe significant improvement in almost all scenarios except for VGG models. The performances of VGG models initially improves in low-budget setting and start to deteriorate quickly when given more budget. We suspect it is due to the fact that all subsampling layers in vanilla VGG are comprised of max-pooling layers. The local maximum remains very similar in discarded activation thus different state provides little additional information.

A2.2 TIMM pretrained-weights on ImageNet

In Fig. A3, we provide additional results on TIMM [62] backbones. These additional backbones include MobileNetV3 [19], Multi-scale ViT (MViTv2) [29], DenseNet [22], VGG [54], RepVGG [12], DeiT [58], CoaT [68], ConvNeXTV2 [35, 64], XCiT [2], VOLO [71], PvTV2 [59, 60], Efficientformer [30]. Specifically, we use these variants from TIMM: `mobilenetv3_small_100` for MobileNetV3, `mvitv2_tiny` for MViTV2, `densenet121` for DenseNet, `vgg11_bn` for VGG, `repvgg_a2` for RepVGG, `deit_tiny_patch16_224` for DeiT, `volo_d1_224` for VOLO, `pvt_v2_b0` for PvTV2, `coat_tiny` for CoaT, `convnextv2_tiny` for CoaT, and `efficientformer_l1` for Efficientformer.

We observe that our approach consistently leads to better performance in most architectures. Our procedure excels in ViT-like architectures where subsampling layers with large subsampling rates R are used.

A2.3 Cityscapes and ADE20K

In Fig. A4 and Fig. A5, we report additional results on Cityscapes and ADE20K semantic segmentation respectively. We plot out the performances on aAcc (mean accuracy of all pixel accuracy) and the mAcc (mean accuracy of each class accuracy) besides the mIoU score. We also provide additional results on more

MMSeg [40] backbones and decoders. These additional architectures include MobileNetV3+LRASPP [19], UNet [49], Swin [34], UperNet [65], and Twins [9]. Specifically, we use these variants from MMSeg: `M-V3-D8` for MobileNetV3, `MiT-B0` for MiT, `Twins-PCPVT-S` for Twins, and `Swin-T` for Swin.

As can be seen, our approach consistently leads to better performance, especially on mIoU and aAcc, in all architectures. Additionally, we show that our test-time procedure can further improve the performance when TTA (`horizontal-flip`) is used.

A2.4 Additional comparison to anti-aliasing downsampling

To preserve the lost information from downsampling layers, a line of work, anti-aliased CNN [73] inserts a `max-blur-pool` layer to the pooling layer. Here, we demonstrate that our approach is orthogonal to the anti-aliasing method.

We report the performance of our approach on anti-aliased CNNs in Tab. A8. We conduct experiments using their pre-trained model and observe that applying our approach to anti-aliased CNNs also leads to consistent performance gain.

Finally, `max-blur-pool` replaces max-pooling layers in CNN and requires training the whole model parameters; it is unclear how to apply it to ViTs which use patch merging instead of max-pooling. Our approach is generic and can be applied to both CNN and ViTs at test time while only training our additional attention module.

Table A8: Performance on the pre-trained weights of anti-aliased CNN [73]. We consider various antialiased CNN backbones with and without our procedure under different $B_{\texttt{ours}} \in \{4, 10\}$. The ✗ indicates the experiments without ours by setting $B_{\texttt{ours}} = 1$. We report the top-1 Acc. on ImageNet. The results w/ our non-learned procedure are highlighted. Our procedure makes improvements on all antialiased CNN backbones.

ResNet18			ResNet34			ResNet50			WideResNet50			MobileNetV2-050		
✗	4	10	✗	4	10	✗	4	10	✗	4	10	✗	4	10
71.67	<u>71.74</u>	**71.88**	74.60	<u>74.75</u>	**74.90**	77.41	<u>77.43</u>	**77.53**	78.70	<u>78.76</u>	**78.91**	72.72	<u>72.98</u>	**73.14**

A3 Additional implementation details

A3.1 Attention module

Recall Eq. (5) and Eq. (6), our attention module contains several trainable modules. Let $\boldsymbol{f_s} = F_\theta(\boldsymbol{I}; \boldsymbol{s}) \in \mathbb{R}^{h\times w\times c}$, then the query, key, and value features are defined as

$$\begin{cases} \boldsymbol{q_s} &= w_q(\boldsymbol{f_s}) \\ \boldsymbol{k_s} &= w_k(\boldsymbol{f_s}) \\ \boldsymbol{v_s} &= \boldsymbol{f_s} \end{cases}, \tag{A15}$$

where $w_{\boldsymbol{q}}$ and $w_{\boldsymbol{k}}$ are both linear layers `Linear(in=c, out=1)`. Finally, the `MLP` in Eq. (6) can be represented by a trainable tensor $w_o \in \mathbb{R}^c$ so that given a input $\boldsymbol{x} \in \mathbb{R}^c$,

$$\texttt{MLP}(\boldsymbol{x}) = w_o \odot \boldsymbol{x}, \tag{A16}$$

where $\odot$ is the elementwise multiplication. Overall, our attention module introduces three learnable parameters, i.e. w_q, w_k, and w_o,

A3.2 TTA policy

Standard policy. The `standard` [52] TTA policy is composed of the following data transformations, `Flip`, `Scale`, and `FiveCrop`. The original `standard` policy fixes $B_{\texttt{tta}}$ at 30. We increase the possible range of $B_{\texttt{tta}}$ for `expanded` policy to cover up to 190. We detail the setting in Tab. A9. For any budget $B_{\texttt{tta}} < 190$, we sorted the data augmentations based on the intensity and sample the first $B_{\texttt{tta}}$ transformations.

Table A9: Details of our modified standard policy. We modified `standard` [52] TTA policy to cover budget up to $B_{\texttt{tta}} = 190$.

Augmentation	# Aug.	Range of p	Description
`FlipLR`	2	`False, True`	Horizontally flip the image if $p =$ `True`.
`Scale`	19	1.00, 1.04, 1.10, 0.98, 0.92, 0.86, 0.80, 0.74, 0.68, 0.62, 0.56, 0.50, 0.44, 0.38, 0.32, 0.26, 0.20, 0.14, 0.08	Scale the image by a ratio of p.
`FiveCrop`	5	`Center`, `LeftTop`, `LeftBottom`, `RightTop`, `RightBottom`	Crop from the p (center or one of the corners) of the image.

Expanded policy. The `expanded` [52] TTA policy contains various data transformations, such as `Saturation`, or `Blur`. The original `expanded` policy fixes $B_{\texttt{tta}}$ at 128. We increase the possible range of $B_{\texttt{tta}}$ for `expanded` policy up to 8778. The pool contains two groups. Each data augmentation of the first group belongs to one of the 131 data augmentations listed in Tab. A10. Each data augmentation of the second group is composed of two of the 131 data augmentations. In total, our pool contains 8778 data augmentation. For any budget $B_{\texttt{tta}} < 8778$, we sorted the data augmentations based on the intensity and sample the first $B_{\texttt{tta}}$ transformations.

Table A10: Details of the our modified expanded policy. We modified `expanded` [52] TTA policy to cover budget up to $B_{\texttt{tta}} = 8778$.

Augmentation	# Aug.	Range of p	Description
`Identity`	1	–	Return the original image.
`FlipLR`	1	–	Horizontally flip the image.
`FlipUD`	1	–	Vertically flip the image.
`Invert`	1	–	Invert the colors of the image.
`PIL_Blur`	1	–	Blur the image using the `PIL.ImageFilter.BLUR` kernel.
`PIL_Smooth`	1	–	Smooth the image using the `PIL.ImageFilter.Smooth` kernel.
`AutoContrast`	1	–	Maximize the contrast of the image.
`Equalize`	1	–	Equalize the histogram of the image.
`Posterize`	4	1, 2, 3, 4	Posterize the image by reducing p RGB bits.
`Rotate`	10	`linspace`(-30, 30, 10)	Rotate the image by p degree.
`CropBilinear`	10	$\{1, 2, \cdots, 10\}$	Crop by (224-p)px and then resize the image.
`Solarize`	10	`linspace`(0, 1, 10)	Solarize the image by a threshold of p.
`Contrast`	10	`linspace`(0.1, 1.9, 10)	Adjust the contrast of the image by a contrast factor of p.
`Saturation`	10	`linspace`(0.1, 1.9, 10)	Adjust the saturation of the image by a saturation factor of p.
`Brightness`	10	`linspace`(0.1, 1.9, 10)	Adjust the brightness of the image by a brightness factor of p.
`Sharpness`	10	`linspace`(0.1, 1.9, 10)	Adjust the sharpness of the image by a sharpness factor of p.
`ShearX`	10	`linspace`(-0.3, 0.3, 10)	Shear the image along the x-axis by $\tan^{-1}(p)$ radians.
`ShearY`	10	`linspace`(-0.3, 0.3, 10)	Shear the image along the y-axis by $\tan^{-1}(p)$ radians.
`TranslateX`	10	`linspace`(-9, 9, 10)	Translate the image along the x-axis by p px.
`TranslateY`	10	`linspace`(-9, 9, 10)	Translate the image along the y-axis by p px.
`Cutout`	10	`linspace`(2, 20, 10)	Randomly mask out p px by p px from the image.

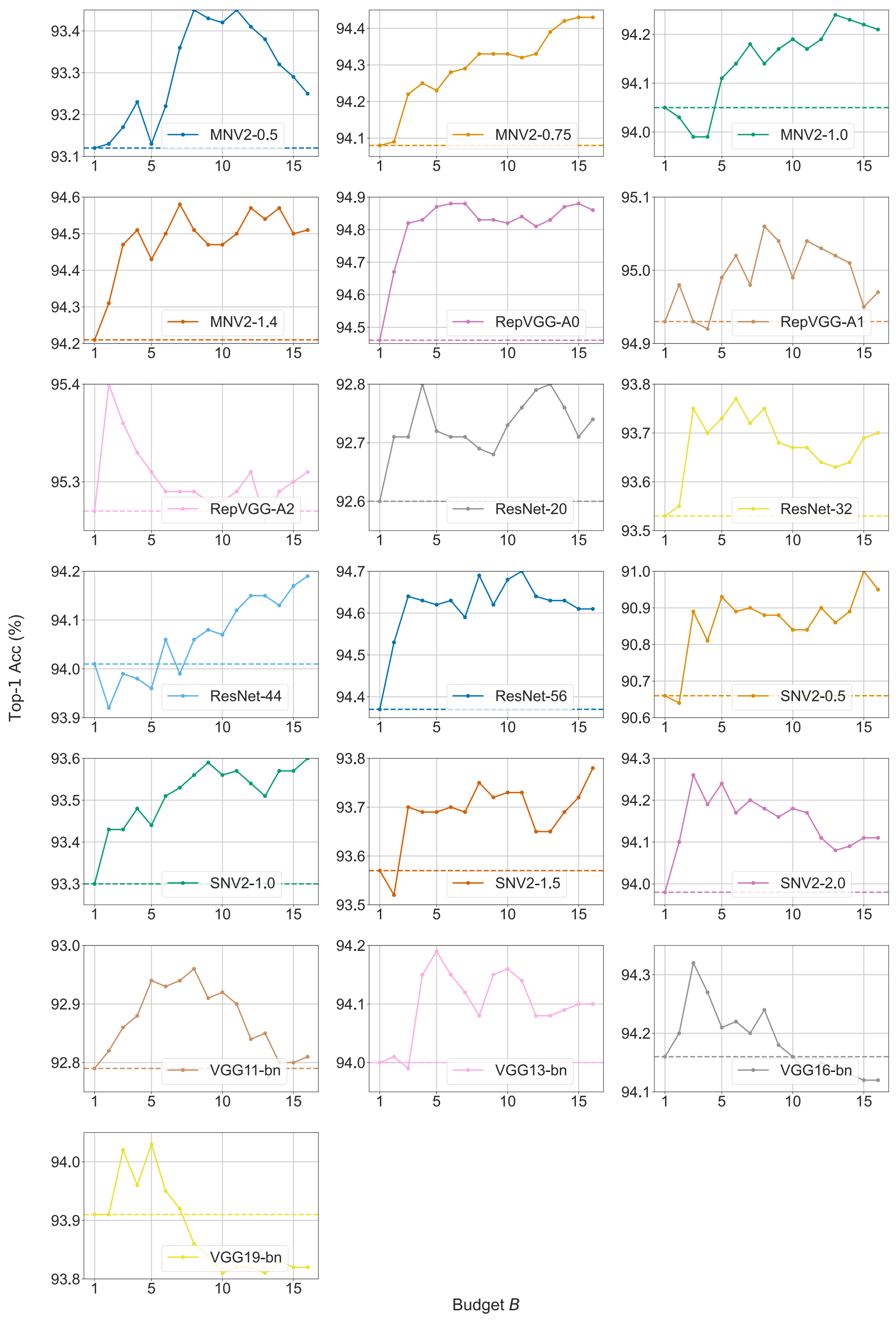

Fig. A1: Results on CIFAR10. We report top-1 accuracy vs. budget $B_{\texttt{ours}}$ used on CIFAR10 image classification. We use the following abbreviation in the figure legends: “MNV2” for MobileNetV2 and “SNV2” for ShuffleNetV2.

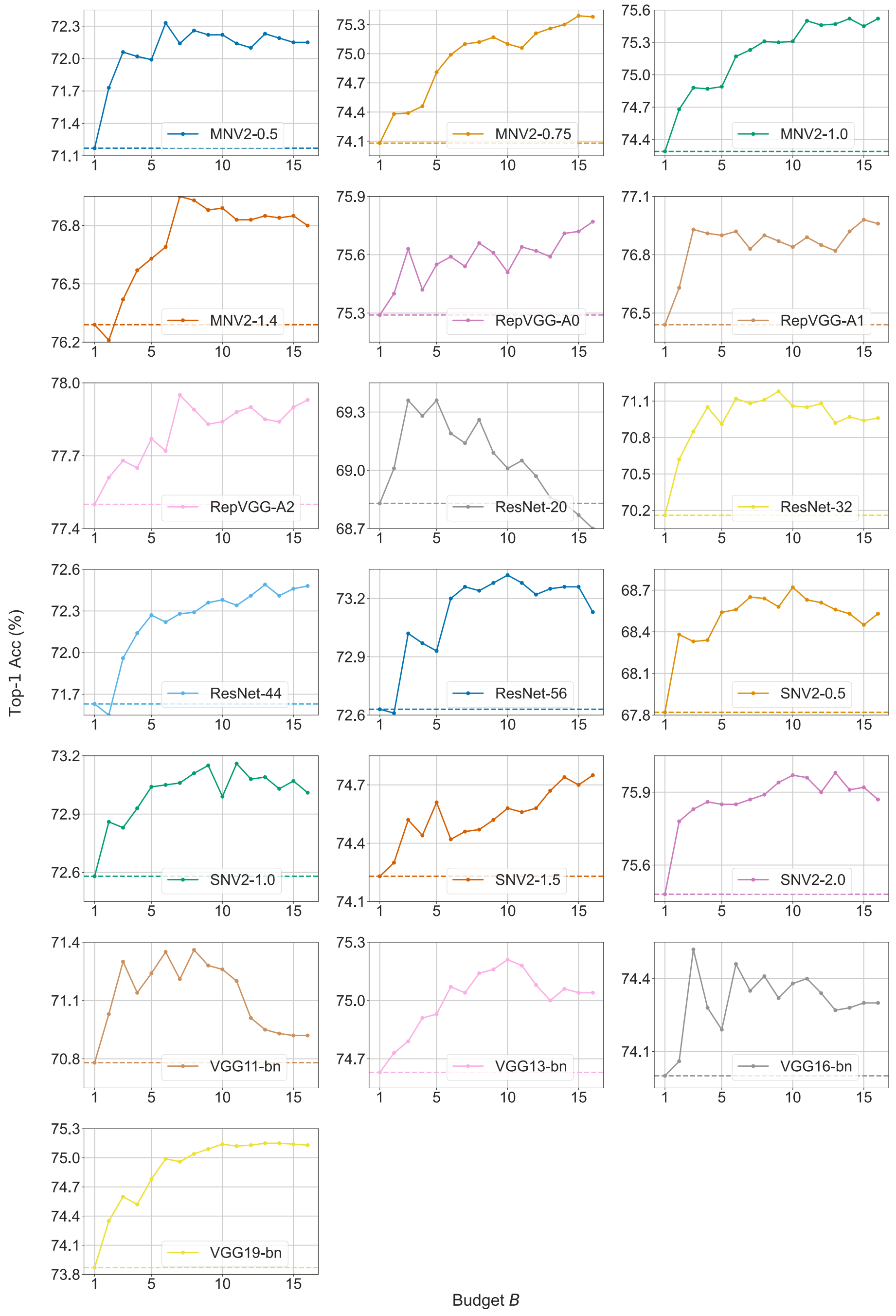

Fig. A2: Results on CIFAR100. We report top-1 accuracy vs. budget $B_{\texttt{ours}}$ used on CIFAR100 image classification. We use the following abbreviation in the figure legends: “MNV2” for MobileNetV2 and “SNV2” for ShuffleNetV2.

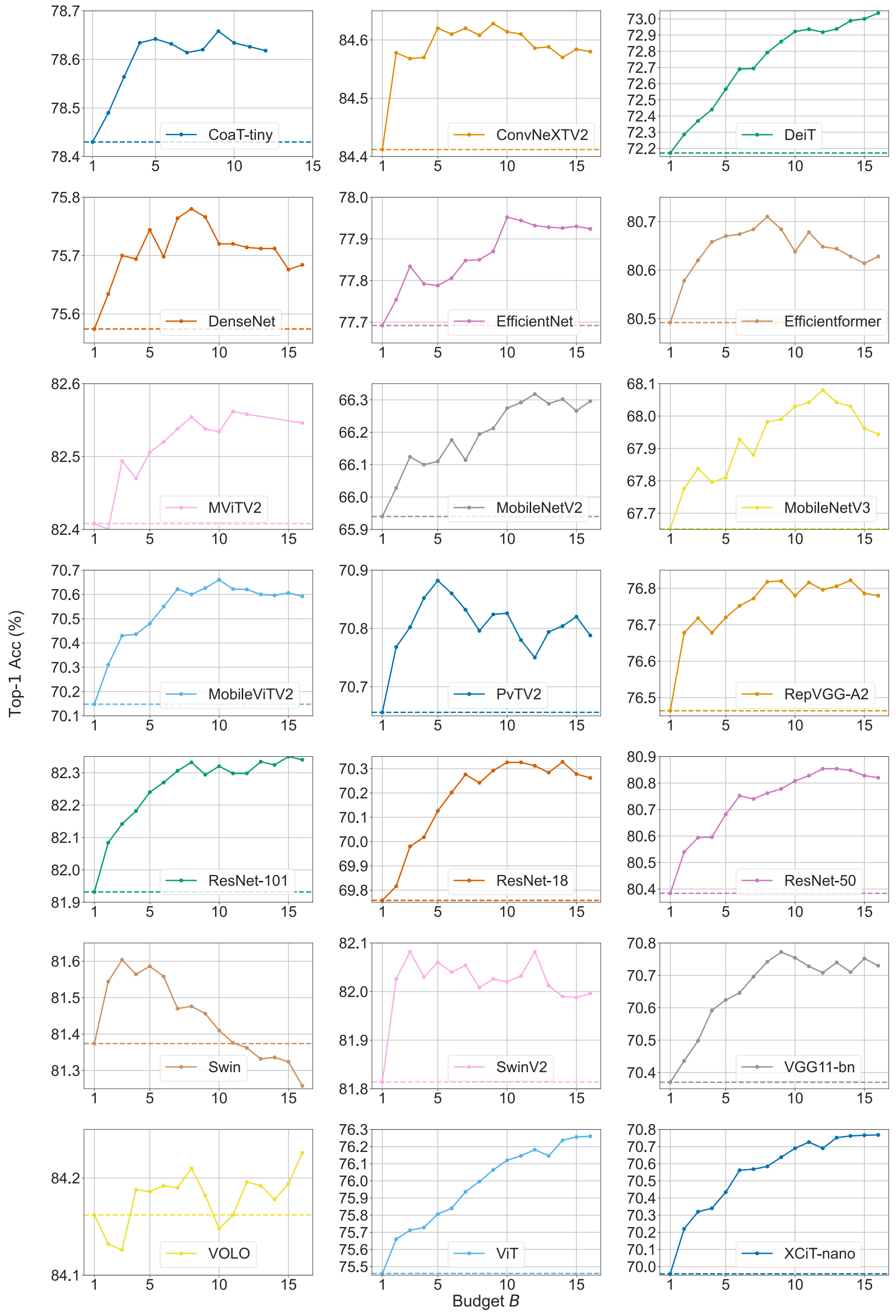

Fig. A3: Additional results on ImageNet. We report top-1 accuracy vs. budget $B_{\texttt{ours}}$ used on ImageNet image classification.

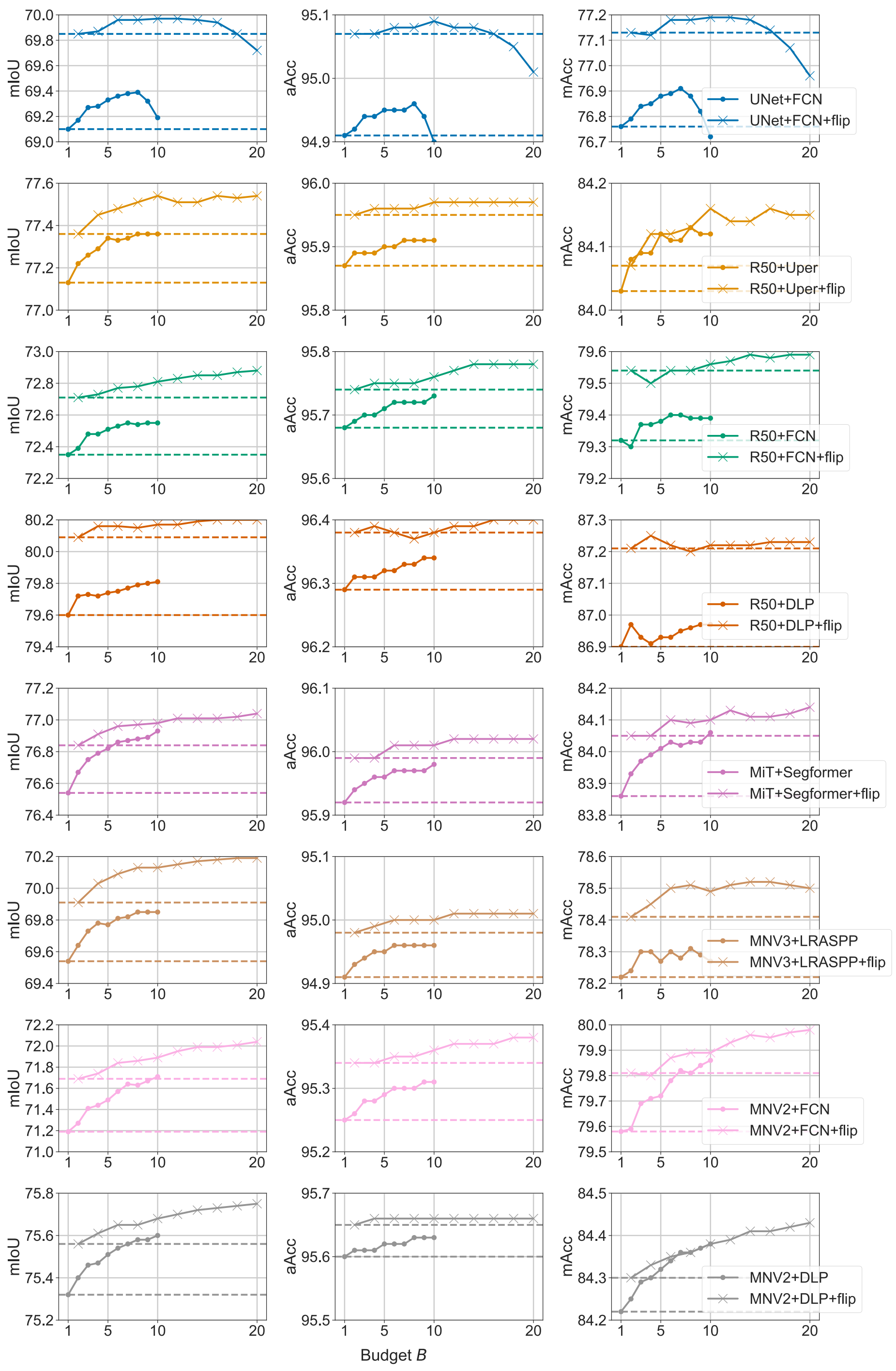

Fig. A4: Additional results on Cityscapes. We report mIoU/aAcc/mAcc vs. budget $B_{\texttt{total}}$ used on Cityscapes semantic segmentation. We note that the $B_{\texttt{total}}$ of "model+flip" is twice the $B_{\texttt{total}}$ of "model".

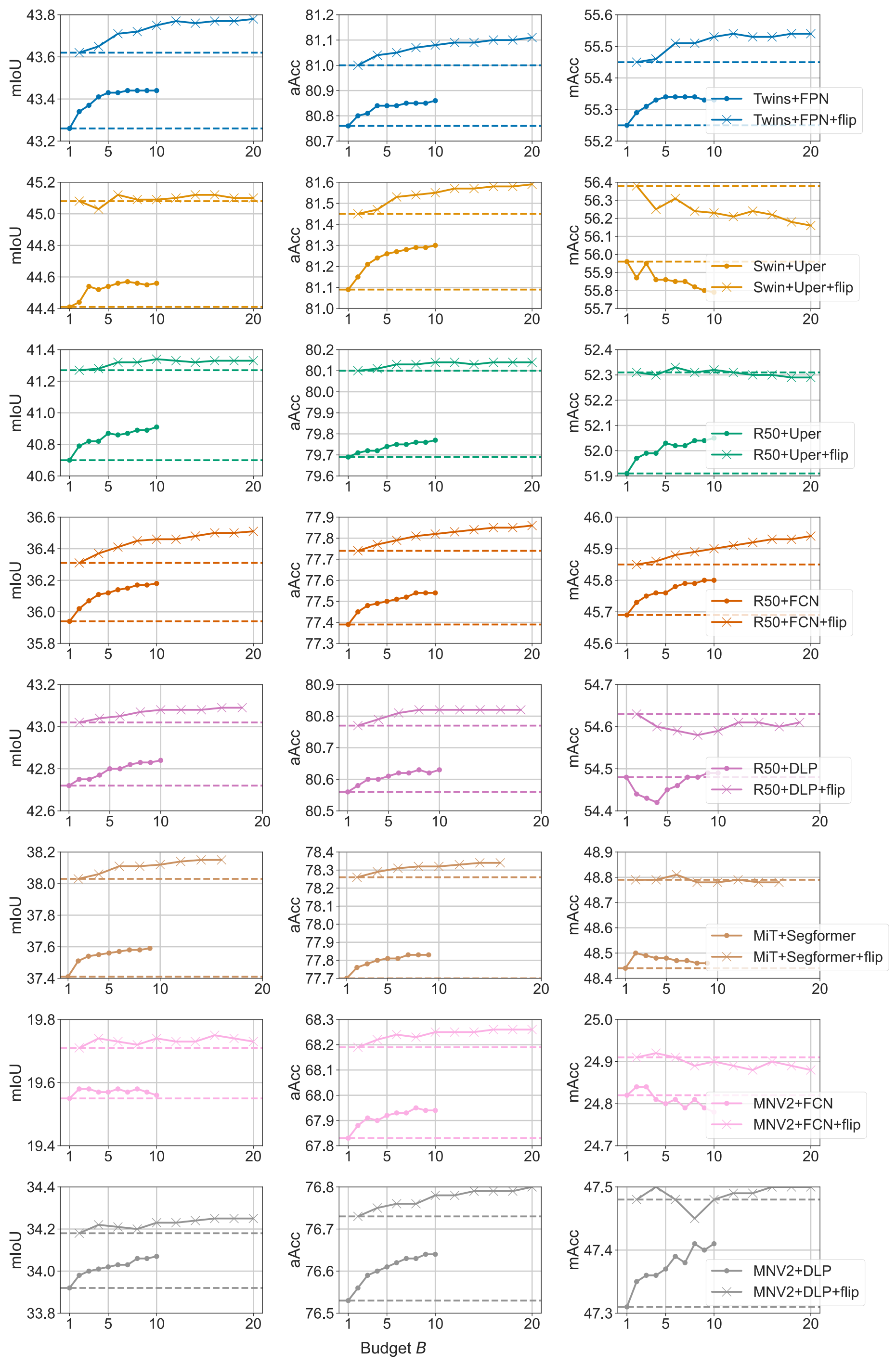

Fig. A5: Additional results on ADE20K. We report mIoU/aAcc/mAcc vs. budget $B_{\texttt{total}}$ used on ADE20K semantic segmentation. We note that the $B_{\texttt{total}}$ of "model+flip" is twice the $B_{\texttt{total}}$ of "model".

A3.3 Computational settings

All experiments are conducted on a single NVIDIA A6000 GPU.

A3.4 Modified subsampling layers

In Fig. A6, we illustrate an implementation trick for faster computation in Alg. 1: line 10. By changing the stride of the subsampling layer from their original stride value R to 1, we can get all the R^2 neighboring states (activations) in a single forward pass.

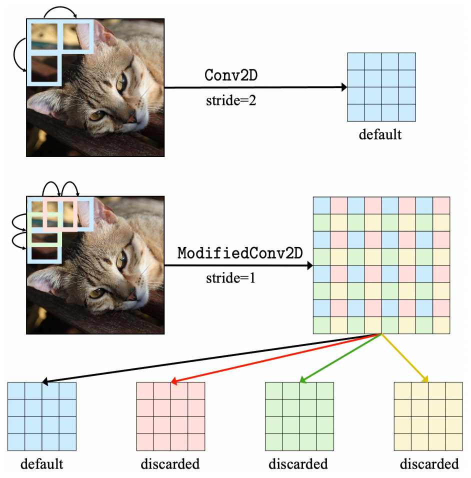

Fig. A6: Stride trick. Instead of performing Conv2D with stride 2 four times, we can equivalently do a stride 1 convolution followed by a pixel unshuffle.

A3.5 Searching procedure

For TIMM implementation, some models, eg. DenseNet and ViT, have fewer (less than 4) subsampling layers. We do not limit the searching space (Alg. 1: line 8) for them in our experiments. Some models, eg. PvTV2, have a larger subsampling rate R (= 4 or 16) on the first subsampling layer than the rest. We do not exclude the first layer from the searching space (Alg. 1: line 8) for them in our experiments.

A3.6 Spatial alignment

For TIMM transformer-based backbones, the output of F_θ has the shape of `batch_size` by `token_length` by `channel_size`. In order to perform our 2-dimensional spatial alignment, we reshape it to `batch_size` by `sqrt(token_length)`

Table A11: Subsampling layers of Torchvision Resnet-18.

Id	Name	Type	R
1	`conv1`	Conv2d	2
2	`maxpool`	MaxPool2d	2
3	`layer2/0`	BasicBlock	2
3-1	`layer2/0/conv1`	Conv2d	2
3-2	`layer2/0/downsample/0`	Conv2d	2
4	`layer3/0`	BasicBlock	2
4-1	`layer3/0/conv1`	Conv2d	2
4-2	`layer3/0/downsample/0`	Conv2d	2
5	`layer4/0`	BasicBlock	2
5-1	`layer4/0/conv1`	Conv2d	2
5-2	`layer4/0/downsample/0`	Conv2d	2

by `sqrt`(`token_length`) by `channel_size` if applicable. For example, the output shape of ViT, XCiT, CoaT, and DeiT features are not composite numbers so we do not reshape them. To aggregate the result from different states, we aggregate on the logits instead. Next, for both the upsampling and downsampling `Resize` in Fig. 3 in the spatial alignment, we use the nearest interpolation.

A3.7 Image classification

In this section, we illustrate how we modify an existing backbone using ResNet-18 as an example. For the Torchvision implementation of ResNet-18, we have listed the subsampling layers in Tab. A11. In our implementation, we modify those layers to include them in the search space. To see how we select these subsampling layers, we have included a code snippet from ``poolsearch/models/cls/torchvision/backbones/resnet.py'' in Fig. A7. For other deep net architectures, please refer to our attached code.

```
def resnet_base(model, num_classes):
    model.num_classes = num_classes
    if model.num_classes != 1000:
        model.fc = nn.Linear(model.feat_dim, model.num_classes)

    member_fns = {}

    def forward_features(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        return x
    member_fns[forward_features.__name__] = forward_features

    def forward_head(self, x):
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
    member_fns[forward_head.__name__] = forward_head

    return model, member_fns

def resnet18_base(model, num_classes):
    model.feat_dim = 512
    model.downsample_layers = [
        ['conv1', 'Conv2d'],
        ['maxpool', 'MaxPool2d'],
        ['layer2/0', 'BasicBlock'],
        ['layer3/0', 'BasicBlock'],
        ['layer4/0', 'BasicBlock'],
    ]
    return resnet_base(model, num_classes)
```

Fig. A7: Code for Torchvision ResNet18 selecting the subsampling layers.