APPENDIX

A HYPERPARAMETER SETTING

We use PyTorch (Paszke et al., 2019) to develop and train our architecture and RobustBench Croce et al. (2021) for the various pre-trained architectures used in the experiments. We use the existing set of pre-trained models (base models) widely used for TTA experiments. We run all the experiments over the NVIDIA A40 GPU. For hyperparameters, we make use of the available set of hyperparameters proposed by the TTA approaches; for example, for using the TTA mechanism proposed by CoTTA, we use hyperparameters provided by CoTTA. For a fair comparison, we use the same optimizers (Adam (Kingma & Ba, 2014) and SGD) as reported by previous TTA baselines. As we decrease the number of parameters by a significant margin, we tune the learning rate for various settings.

For CIFAR10-to-CIFAR10C experiments, we use Adam optimizer (Kingma & Ba, 2014) with a learning rate of 0.00125, and $\beta = 0.9$ with no weight decay. We follow CoTTA for the mean teacher parameter and update the weights of the teacher model by exponential moving average using the student model weights using $\alpha = 0.999$, but without any resetting, i.e., reset rate of 0%.

For CIFAR100-to-CIFAR100C experiments, we use a learning rate of 0.0015 for Adam optimizer with $\beta = 0.9$ and no weight decay. For the mean teacher parameter, we follow CoTTA and update the weights of the teacher model by exponential moving average using the student model weights using $\alpha = 0.999$ with a reset rate of 1%.

In the ImageNet-to-ImageNetC experiments, we use stochastic gradient descent (SGD) optimizer with a learning rate of 0.04, momentum of 0.9, and no weight decay. We follow CoTTA for the mean teacher weight parameter and update teacher model weights by exponential moving average using student model weights with $\alpha = 0.999$, and a reset rate of 0.1%.

The ImageNet-to-ImageNet3DCC experiments use SGD optimizer with a learning rate of 0.03, a momentum of 0.9, with no weight decay. We follow CoTTA for the mean teacher weight parameter. The teacher model weights are updated by exponential moving average, utilizing the student model weights with $\alpha = 0.999$ and a reset rate of 0.1%.

B BENCHMARK DATASETS

Multiple corruptions are introduced to the standard CIFAR10 and CIFAR100 Krizhevsky (2009) datasets to get CIFAR10C and CIFAR100C datasets are corrupted versions, respectively. Both ImageNetC Hendrycks & Dietterich (2019) and ImageNet3DCC Kar et al. (2022) are corrupted versions of the standard ImageNet Deng et al. (2009) dataset.

The CIFAR10C and CIFAR100C datasets both consist of 10,000 images for every corruption type, resulting in a total of 150,000 images for each dataset. The ImageNetC dataset consists of 50,000 images for each corruption class. The CIFAR10C, CIFAR100C, and ImageNet-C datasets comprise a total of 15 distinct types of corruptions, with an additional 4 types designated for validation purposes. Every corruption consists of five distinct levels of severity. The different kinds of corruption, accompanied by concise explanations, are outlined below:

1. Gaussian noise: frequently observed in situations characterized by low illumination levels
2. Shot noise: electrical noise that arises from the discrete character of light
3. Impulse noise: color counterpart of salt-and-pepper noise and can potentially occur owing to bit errors
4. Defocus blur: occurs when an image is captured with an improper focus
5. Frosted glass blur: occurs when an image is seen through a window having frosted glass
6. Motion blur: a phenomenon that arises when a camera undergoes rapid movement
7. Zoom blur: occurs when a camera rapidly moves toward an object
8. Snow: a form of precipitation that visibly obscures the object of interest

9. Frost: happens when ice crystals stick to windows
10. Fog: the presence of fog in the environment causes objects to be obscured from view; this effect is generated using the diamond-square algorithm
11. Brightness: subject to change in accordance with the intensity of sunlight
12. Contrast: influenced by the lighting conditions and the color of the photographed object
13. Elastic transformations: stretching or contracting of small image portions
14. Pixelation: occurs when a low-resolution image is upsampled
15. JPEG: lossy image compression method that leads to the formation of compression artifacts

The ImageNet 3D Common Corruptions (ImageNet3DCC) dataset, as proposed in a recent work by Kar et al. (2022), utilizes the scene geometry for transformations, leading to the generation of corruptions more closely resembling real-world scenarios. The Imagenet3DCC dataset consists of 50,000 images for every form of corruption included within the dataset. It comprises a total of 12 different kinds of corruption, each characterized by 5 degrees of severity. The instances of corruption can be categorized as follows:

1. Near focus: altering the focus region to the nearby section of the scene in a random manner
2. Far focus: introduce random alterations in the focus to encompass the far portion of the scene
3. Bit error: attributed to the presence of imperfections in the video transmission channel
4. Color quantization: reduces the bit depth of an RGB image
5. Flash: occurs when a light source is placed in close proximity to the camera
6. Fog 3D: produced by utilizing a conventional optical model for fog
7. H.265 ABR: H.265 codec in conjunction with the Average Bit Rate control mode for compression purposes
8. H.265 CRF: H.265 codec for compression purposes, specifically employing the Constant Rate Factor (CRF) control mode
9. ISO noise: refers to the presence of noise in an image, which follows a Poisson-Gaussian distribution
10. Low-light: simulated by lowering pixel intensities and addition of Poisson-Gaussian distributed noise
11. XY-motion blur: refers to the blur when the primary camera is in motion along the XY-plane of the picture
12. Z-motion blur: occurs when the primary camera is moving along the Z-axis of the image

The purpose of developing these datasets is to provide standardized benchmarks for evaluating the robustness of classification models.

C Evaluation Metrics

For a given dataset, assume $D = \{x_n, y_n\}_{n=1}^N$, with y_n to be the true label (in one-hot representation, i.e., $y_{ni} = 1$ if i is the true class label, else $y_{ni} = 0$) of x_n, and y'_n to be the prediction by the model.

C.1 Error

The definition of average error rate is as follows:

$$\text{Error} = \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}(y'_n \neq y_n). \tag{1}$$

Here $\mathbb{I}()$ denotes the indicator function.

C.2 Brier Score

The average Brier score Brier (1950) is given by the following:

$$\text{Brier score} = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D}(y'_{ni} - y_{ni})^2. \tag{2}$$

C.3 Negative Log-Likelihood

We define average negative log-likelihood (NLL) as:

$$\text{NLL} = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D}(y_{ni}\log y'_{ni}). \tag{3}$$

D Initialization Scheme Adapter Parameters

We provide a pictorial depiction of the proposed initialization scheme to make the adapters behave as an identity function in the beginning. This ensures that source domain knowledge is intact and, thus, no warm-up using the source data is required, unlike approaches such as Niu et al. (2022); Song et al. (2023).

E Additional Results

For a fair comparison with existing TTA methods, we report all the metrics results corresponding to Table 1 and Table 2 in Table 8 and Table 9, respectively. Overall, we observe that similar performance can be achieved with a significant reduction in trainable parameters.

Comparison with other SOTA methods: The main focus of this work is to design parameter-efficient adapters that are well-suited for lifelong test-time adaptation (TTA). All comparisons conducted in our work are based on the mechanism proposed by CoTTA. In addition, for completeness, we also report a comparative analysis of the performance of our approach with some other recently proposed lifelong TTA approaches.

Table 4 provides a comparison with other existing methods. Since different methods show results on various architectures, we typically use the standard architectures and report the numbers from the paper corresponding to the same architecture and dataset. Note that another recent work EcoTTA (Song et al., 2023) proposes to include meta-network modules to the base model for reducing the activation and memory cost in TTA methods. Adding more parameters in EcoTTA still requires a warm-up phase, making it dependent on the availability of the source dataset. In contrast, our approach `FEATHER` removes this dependency by proposing adapter designs compatible with identity transformation for initialization, making it generic for all TTA methods.

Orthogonality with existing TTA approaches: TENT and EATA only make use of BN parameters in their proposed approach. When compared to the BN parameters adaptable version of both TENT and EATA, TENT+`FEATHER` and EATA+`FEATHER` both achieve significant performance improvement; we speculate the primary reason to be the usage of more parameter space for adaptation without losing the source model weights. Hence, to evaluate the dependence throughout the adaptable parameter space, we create an additional setting in which we update the complete model parameters (100% trainable parameters) and report the findings. Similarly, for comparison with CoTTA, we add an additional setting of adapting only BN parameters. Experimental results in Table 10 show that adapting the entire model parameters does help boost the performance by a significant margin; however, it loses the proxy for source knowledge as all the parameters are now updated. We discover that `FEATHER` can achieve a similar performance improvement with a modest fraction of extra adapter parameters, allowing us to maintain the performance boost with great parameter efficiency without any loss in original model parameters following an update.

In terms of error rate, Table 10 demonstrates that CoTTA with only `FEATHER` adapters being adaptable has an error rate of 34.70%, surpassing CoTTA with only BN params being adaptable, which has an error rate of 32.48%.

Table 8: The table shows all the metrics results obtained for the CIFAR10-to-CIFAR10C online lifelong test-time adaptation task for the highest corruption of severity level 5 corresponding to the results obtained for `FEATHER` reported in Table 1. `FEATHER` here only uses 13.61% adapter parameters compared to the base model. Note that `FEATHER` here, uses the learning objective and TTA scheme proposed by CoTTA.

	Time	$t \longrightarrow$															
Method	**Metric**	*Gaussian*	*shot*	*impulse*	*defocus*	*glass*	*motion*	*zoom*	*snow*	*frost*	*fog*	*brightness*	*contrast*	*elastic*	*pixelate*	*jpeg*	**Mean**
Source	**Error %**	72.33	65.71	72.92	46.94	54.32	34.75	42.02	25.07	41.30	26.01	9.30	46.69	26.59	58.45	30.30	43.51
	Brier	1.29	1.16	1.21	0.79	0.93	0.59	0.71	0.42	0.72	0.44	0.15	0.77	0.44	1.02	0.50	0.74
	NLL	6.46	5.61	5.47	2.74	3.84	2.09	2.51	1.51	3.15	1.53	0.48	2.69	1.38	4.67	1.65	3.05
BN	**Error %**	28.08	26.12	36.27	12.82	35.28	14.17	12.13	17.28	17.39	15.26	8.39	12.63	23.76	19.66	27.30	20.44
	Brier	0.46	0.43	0.59	0.20	0.57	0.23	0.19	0.28	0.28	0.24	0.13	0.20	0.38	0.32	0.45	0.33
	NLL	1.46	1.32	1.90	0.57	1.76	0.64	0.54	0.82	0.82	0.71	0.36	0.57	1.14	0.92	1.38	0.99
TENT	**Error %**	24.80	20.48	28.49	14.84	31.78	16.97	16.66	21.97	20.97	20.92	14.76	19.91	27.56	23.89	31.01	22.33
	Brier	0.42	0.35	0.50	0.26	0.56	0.30	0.30	0.40	0.39	0.38	0.27	0.37	0.51	0.44	0.58	0.40
	NLL	1.41	1.33	2.10	1.11	2.61	1.52	1.65	2.34	2.43	2.42	1.76	2.48	3.21	2.97	4.17	2.23
CoTTA	**Error %**	23.92	21.40	25.95	11.82	27.28	12.56	10.48	15.31	14.24	13.16	7.69	11.00	18.58	13.83	17.17	16.29
	Brier	0.36	0.33	0.38	0.18	0.40	0.19	0.16	0.23	0.21	0.20	0.11	0.16	0.27	0.20	0.25	0.24
	NLL	0.92	0.85	0.88	0.43	0.93	0.46	0.37	0.55	0.50	0.46	0.26	0.36	0.60	0.45	0.56	0.57
FEATHER	**Error %**	24.76	21.98	26.82	11.92	28.33	12.55	10.62	15.28	14.41	13.26	7.77	12.03	19.39	14.49	18.17	16.79
	Brier	0.38	0.34	0.39	0.18	0.42	0.19	0.16	0.23	0.21	0.20	0.11	0.17	0.28	0.21	0.27	0.25
	NLL	1.02	0.92	0.92	0.44	0.98	0.45	0.37	0.54	0.50	0.45	0.24	0.39	0.62	0.47	0.60	0.59

Table 9: The table shows all the metrics results obtained for the CIFAR100-to-CIFAR100C online lifelong test-time adaptation task for the highest corruption of severity level 5 corresponding to the results obtained for `FEATHER` reported in Table 2. `FEATHER` here only uses 6.8% adapter parameters compared to the base model. Note that `FEATHER` here, uses the learning objective and TTA scheme proposed by CoTTA.

	Time	$t \longrightarrow$															
Method	**Metric**	*Gaussian*	*shot*	*impulse*	*defocus*	*glass*	*motion*	*zoom*	*snow*	*frost*	*fog*	*brightness*	*contrast*	*elastic*	*pixelate*	*jpeg*	**Mean**
Source	**Error %**	73.00	68.01	39.37	29.32	54.11	30.81	28.76	39.49	45.81	50.30	29.53	55.10	37.23	74.69	41.25	46.45
	Brier	1.11	1.04	0.58	0.41	0.79	0.43	0.40	0.53	0.64	0.71	0.41	0.75	0.51	1.12	0.56	0.67
	NLL	5.59	4.89	2.00	1.19	2.86	1.26	1.16	1.63	2.12	2.34	1.16	2.52	1.50	5.39	1.74	2.49
BN	**Error %**	42.14	40.66	42.73	27.64	41.82	29.72	27.87	34.88	35.03	41.50	26.52	30.31	35.66	32.94	41.16	35.37
	Brier	0.55	0.54	0.56	0.37	0.55	0.40	0.38	0.47	0.46	0.55	0.36	0.40	0.48	0.44	0.54	0.47
	NLL	1.69	1.62	1.71	1.06	1.64	1.13	1.06	1.38	1.37	1.66	1.01	1.17	1.40	1.29	1.66	1.39
TENT	**Error %**	37.16	35.61	41.82	37.54	51.19	48.48	49.15	58.83	62.85	71.65	70.76	82.91	88.00	91.14	94.63	61.45
	Brier	0.51	0.52	0.63	0.60	0.82	0.82	0.8585	1.03	1.13	1.31	1.32	1.60	1.68	1.77	1.85	1.10
	NLL	1.49	1.58	2.14	2.12	3.28	3.66	4.17	5.46	6.71	8.53	9.04	14.43	14.17	16.21	17.66	7.37
CoTTA	**Error %**	40.09	37.67	39.77	26.91	37.82	28.04	26.26	32.93	31.72	40.48	24.72	26.98	32.33	28.08	33.46	32.48
	Brier	0.53	0.51	0.53	0.37	0.50	0.38	0.36	0.44	0.43	0.53	0.35	0.37	0.44	0.39	0.45	0.44
	NLL	1.60	1.50	1.58	1.04	1.47	1.09	1.02	1.29	1.24	1.59	0.96	1.05	1.25	1.10	1.30	1.27
FEATHER	**Error %**	40.10	36.66	38.81	26.68	38.10	28.56	25.95	33.81	32.42	42.12	24.98	27.32	34.31	28.60	35.40	32.92
	Brier	0.53	0.5	0.52	0.37	0.51	0.39	0.36	0.46	0.44	0.55	0.35	0.38	0.46	0.39	0.47	0.45
	NLL	1.6	1.46	1.51	1	1.46	1.07	0.97	1.3	1.22	1.65	0.93	1.03	1.28	1.07	1.34	1.26

Flexibility in Parameter Efficiency: We report the exact number of parameters for Table 5 in the Table 11. This illustrates the flexibility of `FEATHER` to choose the number of parameters depending on the memory and computational budget.

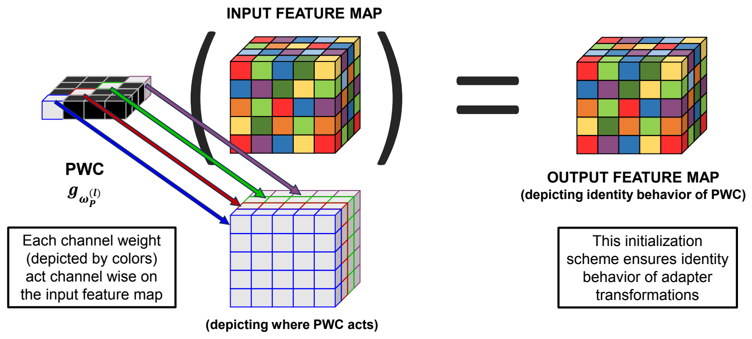
INPUT FEATURE MAP
PWC
$g_{\omega_P^{(l)}}$
Each channel weight (depicted by colors) act channel wise on the input feature map
(depicting where PWC acts)
OUTPUT FEATURE MAP
(depicting identity behavior of PWC)
This initialization scheme ensures identity behavior of adapter transformations

Figure 3: The figure highlights the proposed initialization scheme for the added adapter parameters. Specifically, the Point Wise Convolution (PWC), when initialized with diagonal elements (highlighted in white), acts channelwise on the input feature map, making the interaction between the channels zero and projecting the same feature space to act as the output feature map.

Table 10: The table compares FEATHER applied over TENT and CoTTA, highlighting the orthogonality of the proposed generic framework. Results for the CIFAR100-to-CIFAR100C benchmark depict the parameter efficiency obtained (**93.2% fewer parameters**) with a meager performance drop (0.71% for TENT (100% params), 0.52% for EATA (100% params), and 0.44% for CoTTA).

Time	t →															
Method	***Gaussian***	***shot***	***impulse***	***defocus***	***glass***	***motion***	***zoom***	***snow***	***frost***	***fog***	***brightness***	***contrast***	***elastic***	***pixelate***	***jpeg***	**Mean**
TENT-lifelong	**37.20**	35.80	41.70	37.90	51.20	48.30	48.50	58.40	63.70	71.10	70.40	82.30	88.00	88.50	90.40	60.90
TENT-lifelong (100% Params)	40.27	**35.68**	**37.28**	**26.27**	**37.81**	28.98	26.97	**33.39**	**32.52**	**39.21**	27.33	32.39	34.87	32.03	**39.65**	**33.64**
+ FEATHER (6.8% Params)	41.67	39.40	41.35	26.96	40.11	**28.87**	**26.91**	33.87	33.80	39.94	**26.27**	**29.55**	**34.50**	**32.02**	40.10	34.35
EATA-lifelong	41.83	40.27	42.56	27.56	41.54	29.54	27.70	34.69	34.71	41.24	26.42	30.2	35.58	32.73	40.95	35.17
EATA-lifelong (100% Params)	41.50	**38.81**	**41.07**	**26.82**	**39.90**	**28.90**	26.83	**33.56**	**33.11**	**39.60**	**25.38**	**29.00**	**34.15**	**30.79**	**38.96**	**33.89**
+ FEATHER (6.8% Params)	**41.21**	38.96	41.24	26.97	41.07	29.26	27.13	33.84	34.3	39.94	26.04	30.14	34.61	31.67	39.74	34.41
CoTTA (BN Params)	40.33	38.3	40.16	27.67	39.99	29.76	27.88	35.51	34.68	43.4	26.58	30.52	35.98	32.05	37.71	34.70
CoTTA (100% Params)	**40.09**	37.67	39.77	26.91	**37.82**	**28.04**	26.26	**32.93**	**31.72**	**40.48**	**24.72**	**26.98**	**32.33**	**28.08**	**33.46**	**32.48**
+ FEATHER (6.8% Params)	40.10	**36.66**	**38.81**	**26.68**	38.10	28.56	**25.95**	33.81	32.42	42.12	24.98	27.32	34.31	28.60	35.40	32.92

Table 11: Error rate (%) on CIFAR100C over different percentages of added parameters in FEATHER. The (Parameter %) in the bracket in the first two columns indicates a comparison with the base model. For example, 494208 (7.16% of 6900132) and 7369124 (106.80% of 6900132). We observe that adding a similar number of trainable parameters as adapters (101.29%, last row) improves the performance over CoTTA by a small margin, and even with a much smaller number of trainable parameters, FEATHER achieves comparable performance.

Method	**Trainable params**	**Total Params**	**Trainable %**	**CIFAR100C**
CoTTA	6900132	6900132	100.00%	32.5
FEATHER	494208 (7.16%)	7369124 (106.80%)	**6.71%**	32.92
	2354816 (34.12%)	9229732 (133.76%)	25.51%	32.79
	3943040 (57.14%)	10817956 (156.78%)	36.45%	32.65
	6988800 (101.29%)	13888932 (201.29%)	50.32%	**32.31**

F ARCHITECTURE DETAILS ALONG WITH FEATHER ADAPTERS

In this section, we report the architecture-specific details used for adapter parameters. For CIFAR100-to-CIFAR100C experiments, we modify the widely used ResNeXt-29 architecture taken from RobustBench Croce et al. (2021) and added adapters in between referred to as ConvAdapt layers. For ImageNet-to-ImageNetC, we modify the ResNet-50 architecture and add adapter layers in between. The added adapter layers are kept in bold.

ResNeXt-29 with adapters for CIFAR100-to-CIFAR100C

```
Hendrycks2020AugMixResNeXtNetAdpt(
        (conv_1_3x3): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn_1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
  (stage_1): Sequential(
    (0): ResNeXtBottleneck(
      (conv_reduce): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_reduce): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_conv): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=4, bias=False)
      (bn): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_expand): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_expand): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      )
    )
    (1): ResNeXtBottleneck(
      (conv_reduce): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_reduce): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_conv): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=4, bias=False)
      (bn): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_expand): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_expand): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    )
    (2): ResNeXtBottleneck(
      (conv_reduce): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_reduce): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_conv): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=4, bias=False)
      (bn): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_expand): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_expand): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    )
  )
  (stage_2): Sequential(
    (0): ResNeXtBottleneck(
      (conv_reduce): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_reduce): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=4, bias=False)
      (bn): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_expand): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_expand): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      )
    )
    (1): ResNeXtBottleneckAdpt(
      (conv_reduce): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_reduce): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (lhc1): ConvAdapt(
        (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
        (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      )
      (conv_conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=4, bias=False)
      (bn): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (lhc2): ConvAdapt(
        (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
        (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      )
      (conv_expand): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_expand): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (lhc3): ConvAdapt(
        (gwc): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=64)
        (pwc): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
      )
    )
    (2): ResNeXtBottleneck(
      (conv_reduce): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_reduce): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=4, bias=False)
      (bn): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_expand): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_expand): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    )
  )
  (stage_3): Sequential(
    (0): ResNeXtBottleneck(
      (conv_reduce): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_reduce): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_conv): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=4, bias=False)
      (bn): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_expand): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_expand): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      )
    )
    (1): ResNeXtBottleneck(
      (conv_reduce): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_reduce): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_conv): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=4, bias=False)
      (bn): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_expand): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_expand): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    )
    (2): ResNeXtBottleneck(
      (conv_reduce): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_reduce): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_conv): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=4, bias=False)
      (bn): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (conv_expand): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn_expand): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (classifier): Linear(in_features=1024, out_features=100, bias=True)
)
```

ResNet-50 with adapters for ImageNet-to-ImageNetC

```
ResNetAdapt(
    (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): BottleneckAdpt(
        (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=8)
          (pwc): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=8)
          (pwc): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        )
      )
      (1): BottleneckAdpt(
        (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=8)
          (pwc): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=8)
          (pwc): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
      )
      (2): BottleneckAdpt(
        (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=8)
          (pwc): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=8)
          (pwc): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
      )
    )
    (layer2): Sequential(
      (0): BottleneckAdpt(
        (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=16)
          (pwc): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=16)
          (pwc): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        )
      )
      (1): BottleneckAdpt(
        (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=16)
          (pwc): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=16)
          (pwc): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
      )
      (2): BottleneckAdpt(
        (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=16)
          (pwc): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=16)
          (pwc): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
      )
      (3): BottleneckAdpt(
        (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=16)
          (pwc): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=16)
          (pwc): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
      )
    )
    (layer3): Sequential(
      (0): BottleneckAdpt(
        (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
          (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
          (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        )
      )
      (1): BottleneckAdpt(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
          (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
          (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
      )
      (2): BottleneckAdpt(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
          (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
          (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
      )
      (3): BottleneckAdpt(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
          (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
          (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
      )
      (4): BottleneckAdpt(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
          (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
          (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
      )
      (5): BottleneckAdpt(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
          (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
          (pwc): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
```

```
        (relu): ReLU(inplace=True)
      )
    )
    (layer4): Sequential(
      (0): BottleneckAdpt(
        (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=64)
          (pwc): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=64)
          (pwc): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        )
      )
      (1): BottleneckAdpt(
        (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=64)
          (pwc): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=64)
          (pwc): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
      )
      (2): BottleneckAdpt(
        (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (lhc1): ConvAdapt(
          (gwc): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=64)
          (pwc): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (lhc2): ConvAdapt(
          (gwc): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=64)
          (pwc): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
        )
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
        (relu): ReLU(inplace=True)
      )
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Linear(in_features=2048, out_features=1000, bias=True)
  )
)
```