A. Implementation Details

We describe here the architectures, hyperparameters, and other implementation details used in our experiments.

A.1. Class Incremental Learning

CIFAR-100 [19] is a medium-scale dataset widely used for supervised classification. It contains 100 diverse classes that can grouped into 20 superclasses. We leverage the CIFAR-100 dataset for the class incremental learning setup. We perform experiments in three settings: CIFAR-100/10 which contains 10 task of 10 classes each, CIFAR-100/20 which contains 20 task of 5 classes each and CIFAR-100/5 which contains 5 task of 20 class each. CIFAR-100/20 tests the model's ability to resist catastrophic forgetting for a large number of task, while CIFAR-100/5 tests the model's ability when each task contains more classes. Interestingly, we observe that many models from the literature showing good results for a small number of tasks perform badly for a larger number of tasks. Thus, we find a large number of tasks to be a more reliable setting to test a model's continual learning ability.

We use the ResNet-18 [11] architecture for all experiments on the CIFAR dataset. ResNet-18 contains 11.69M parameters across 18 layers. The proposed *Efficient Feature Transformation* (EFT) module can be applied in two ways: *i*) Serial adaptation and *ii*) parallel adaptation (see Section 2.1.3). The added serial adapter on ResNet-18 architecture is shown in the Figure 6 (top).

We use the SGD optimizer for all experiments with a weight decay of 0.005. The ResNet-18 architecture is trained for 250 epochs for the first task and 200 epochs for subsequent tasks, as forward transfer from the first task leads to quicker convergence. We train the first task with an initial learning rate of 0.01, with $0.1\times$ learning rate decay at epochs 100, 150, and 200. For the second task onward, the same initial learning and decay rate are used, with decay steps occurring at epochs 70, 120 and 150. In all our experiments (CIFAR-100/10, CIFAR-100/20 and CIFAR-100/5), we set $\lambda = 0.05$. For hyperparameter tuning, 10% data was held out as validation data; for final training, we merge the training data and validation data. In Equation 4, $\gamma = 0$ indicates that the 1×1 filters are not used for the EFT transformation. We show a setting with such a case in Tables 6 and 8. In the class incremental learning scenario, task IDs are available during training, but they must be inferred at test time to determine the corresponding τ_t. During training, only samples of task t are present, while at test time, test samples may come from any of tasks 1 to t.

A.2. Task Incremental Learning

We perform experiments in the task incremental learning scenario on medium- and large-scale datasets with two standard architectures: VGG-16 [47] and AlexNet [20].

ImageNet-1K The ImageNet-1K [6] classification dataset contains 1000 classes based on the WordNet [33] hierarchy. In the continual learning setup, 1000 classes are divided into 10 tasks of 100 classes each. We use the SGD optimizer to optimize an AlexNet architecture. The model is trained for 90 epochs for ImageNet with a weight decay of 0.0005 for the first task and a weight decay of 0.00005 for subsequent tasks. An initial learning rate of 0.0001 is used with a step decay of $0.5\times$ at epochs 40, 60 and 80. EFT layers are appended after each layer.

Tiny-ImageNet-200 The Tiny-ImageNet-200 dataset is a subset of the ImageNet dataset and contains 200 classes at downsampled resolution. For continual learning, these 200 classes are divided into 10 tasks with 20 classes each. Since Tiny ImageNet has a lower resolution, the last maxpool layer and the last three convolutional layers from the feature extractor are removed from the standard VGG-16 architecture [30]. The VGG-16 model is trained for 160 epochs with an initial learning rate of 0.01 and a weight decay of 0.0005, for all tasks. The learning rate is decayed by a factor of $0.1\times$ at epochs 70, 100, and 120. EFT layers are appended as shown in Figure 6.

A.3. VGG-16 for Heterogeneous Datasets

Many previous approaches evaluate the model's performance in homogeneous settings: tasks strongly resemble the preceding ones. We evaluate the model on heterogeneous datasets, where later tasks are very different from previous ones, and the number of classes may change between tasks. In this challenging setup, we use the VGG-16 architecture with batchnorm to learn CIFAR-10, SVHN, and CIFAR-100 in sequence. We show results for two different task orders: CIFAR-100→CIFAR-10→SVHN and SVHN→CIFAR-10→CIFAR-100. We report the results in Table 3 as the final performance of the model on each of the datasets at the conclusion of the task sequence. The VGG-16 architecture is trained for 200 epochs with an initial learning rate of 0.01 and a weight decay of 0.0005. A learning rate decay of $0.1\times$ is used at steps 100, 150, and 170.

A.4. Parallel Adaptation

An alternative parallel adaptation strategy is proposed in Section 2.1.3. We show results for the parallel

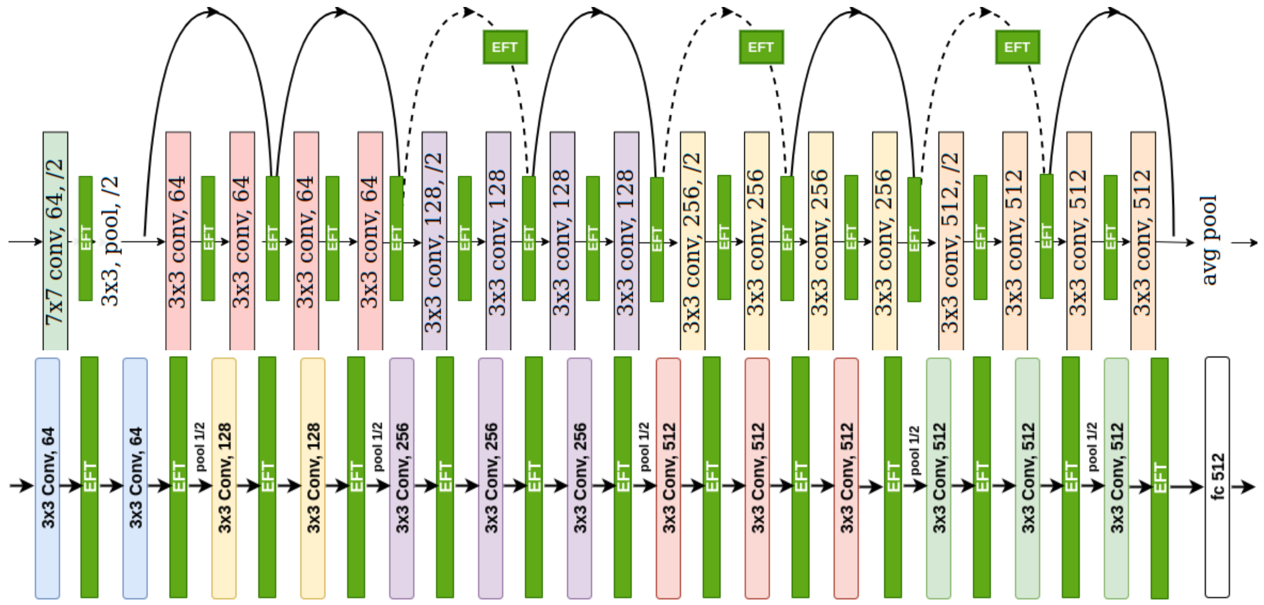
Figure 6. Overview of inserted transformations for ResNet-18. The added EFT layers in each architecture used for continual learning are shown in green. Except for the EFTs, all parameters remain unchanged between tasks.

adaptation strategy in Table 8. We observe that parallel adaptation with EFTs results in slightly inferior performance compared to serial adaptation. Note that for both parallel and serial adaptation, we use the same optimizer, learning rate, learning rate decay, and weight decay. Please refer to Section A.1 for more details.

A.5. StackGAN-v2 for Generative Modeling

Continual learning in generative models (*e.g.*, GANs) is a challenging problem rarely explored by recent literature. Existing methods tend to rely on replay-based approaches. In contrast, we use an expansion-based method to learn to generate successive datasets in a continual fashion. Specifically, we use StackGAN-v2 [64] as the base network and use EFT layers to continually expand the architecture for each novel task. We append EFT layers after each convolutional layer (serial adaptation) in both the generator and the discriminator, before batchnorm and ReLU activations are applied, resulting in 4.8% extra parameters per dataset. We use the default StackGAN-v2 hyperparameter values for learning rate and the number of epochs.

B. Others Comparisons

The model architectures used for continual learning tend to vary from paper to paper, so we did our best to compare our proposed approach with recent baselines in a fair, consistent setting. However, other papers occasionally report results with other setups. For example, continual learning performance on CIFAR is occasionally reported with CIFAR-10 being the base dataset,

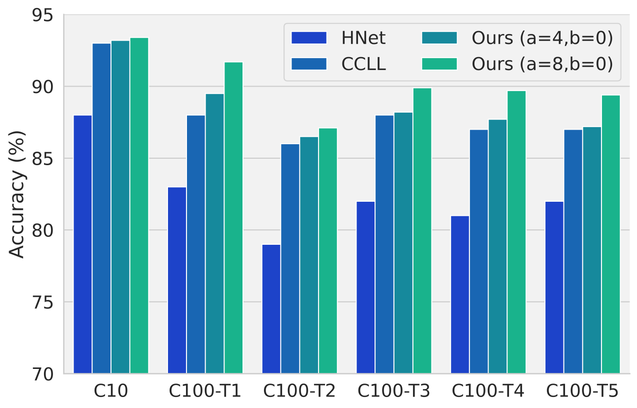

Figure 7. Comparison of the proposed model on the ResNet-32 architecture on the CIFAR10-CIFAR100 dataset.

with 5 subsequent tasks of 10 classes drawn from the CIFAR-100 dataset. For this setting, we follow prior work in using ResNet-32 [11], an architecture with a smaller number of parameters and FLOPs commonly used for CIFAR. We compare our proposed approach with the recent works HNet [51] and CCLL [48] in this setting. The results are shown in Figure 7. We use the same training procedure as discussed we did with the ResNet-18 architecture. We see stronger results with our EFT approach.

ResNet-18/3 [62], a standard ResNet-18 but with a third of the filters per layer, is another widely used architecture for continual learning. We compare our EFTs with the recently proposed SupSup [54] and BatchE [53] on CIFAR-100/20 in the task incremental setting, showing the results in Table 9. Again, we observe the pro-

Table 8. Average accuracy on CIFAR-100 in class incremental learning setting when trained on 10 tasks sequentially.

Dataset / #Tasks	Methods	1	2	3	4	5	6	7	8	9	Final
CIFAR-100/10	Finetune	88.5	47.1	32.1	24.9	20.3	17.5	15.4	13.5	12.5	11.4
	FixedRep	88.5	45.9	30.1	22.4	17.7	15.2	12.3	11.1	9.8	8.8
	LwF [25]	88.5	70.1	54.8	45.7	39.4	36.3	31.4	28.9	25.5	23.9
	EWC [18]	88.5	52.4	48.6	38.4	31.1	26.4	21.6	19.9	18.8	16.4
	EWC+SDC [61]	88.5	78.8	75.8	73.1	71.5	60.7	53.9	43.5	29.5	19.3
	SI [62]	88.5	52.9	40.7	33.6	31.8	29.4	27.5	25.6	24.7	23.3
	MAS [1]	88.5	42.1	36.4	35.1	32.5	25.7	21.0	19.2	17.7	15.4
	RWalk [2]	88.5	55.1	40.7	32.1	29.2	25.8	23.0	20.7	19.5	17.9
	DMC [65]	88.5	76.3	67.5	62.4	57.3	52.7	48.7	43.9	40.1	36.2
	EFT-a_4b_0 (+1.7%)	90.3	73.9	65.9	59.0	54.6	50.6	48.4	45.6	43.3	**41.2**
	EFT-a_4b_8 (+2.0%)	90.3	73.9	65.9	59.0	54.6	50.6	48.4	45.6	43.3	**41.5**
	EFT-a_8b_{16} (+3.9%)	90.3	74.2	66.5	60.4	55.8	51.5	49.5	46.7	44.7	**42.7**

posed model shows significant improvements compared to the baseline model.

Entry	Avg Acc@1
Upper Bound	91.62 ± 0.89
SupSup (GG) [54]	86.45 ± 0.61
SupSup (GG) Transfer [54]	88.52 ± 0.85
BatchE (GG) [53]	79.75 ± 1.00
Separate Heads	70.60 ± 1.40
EFT a_4b_0(+5.2%)	89.25 ± 0.31
EFT a_5b_0(+6.7%)	90.17 ± 0.47

Table 9. Task incremental learning accuracy on CIFAR-100/20 with the ResNet-18/3 [62] architecture.

C. Selection of Hyperparameter a and b

The choice of the hyperparameters a and b control both the model's representational power per task, as well as the growth in parameters and FLOPs. This leads to a trade-off: increasing the value of a and b will increase the model's performance at the cost of higher computation and memory (refer to Table-6 for ablation). Optimal values depend on the situation and can vary based on architecture. We observe that for ResNet [11] and VGG [47], we can use lower values a and b, leading to a parameter growth of 2-4%, while for the AlexNet [20] architecture, higher values of a and b only lead to 0.6% growth, as a significant proportion of the model parameters are in the fully connected layer. Using lower hyperparameter values for AlexNet results in a very small number of parameters per task, making it difficult for the model to adapt to novel tasks. Additionally, the proposed EFT for fully connected layers only adds a diagonal matrix for continual adaption, which adds only a negligible number of parameters. We observe that architectures with more parameters in fully connected layers (especially AlexNet) have a harder time adapting features with just diagonal matrices. Therefore, we find it advantageous to increase the number of convolutional layer parameters. We can observe this in Table 2 for the AlexNet architecture.