# CLR: Channel-wise Lightweight Reprogramming for Continual Learning

Yunhao Ge[1†], Yuecheng Li[1*], Shuo Ni[1*], Jiaping Zhao[2] Ming-Hsuan Yang[2], Laurent Itti[1†]
[1]University of Southern California [2]Google Research
[*]Equal contribution as second author, [†]correspondence to {yunhaoge, itti}@usc.edu

https://github.com/gyhandy/Channel-wise-Lightweight-Reprogramming

## Abstract

*Continual learning aims to emulate the human ability to continually accumulate knowledge over sequential tasks. The main challenge is to maintain performance on previously learned tasks after learning new tasks, i.e., to avoid catastrophic forgetting. We propose a Channel-wise Lightweight Reprogramming (CLR) approach that helps convolutional neural networks (CNNs) overcome catastrophic forgetting during continual learning. We show that a CNN model trained on an old task (or self-supervised proxy task) could be "reprogrammed" to solve a new task by using our proposed lightweight (very cheap) reprogramming parameter. With the help of CLR, we have a better stability-plasticity trade-off to solve continual learning problems: To maintain* ***stability*** *and retain previous task ability, we use a common task-agnostic immutable part as the shared "anchor" parameter set. We then add task-specific lightweight reprogramming parameters to reinterpret the outputs of the immutable parts, to enable* ***plasticity*** *and integrate new knowledge. To learn sequential tasks, we only train the lightweight reprogramming parameters to learn each new task. Reprogramming parameters are task-specific and exclusive to each task, which makes our method immune to catastrophic forgetting. To minimize the parameter requirement of reprogramming to learn new tasks, we make reprogramming lightweight by only adjusting essential kernels and learning channel-wise linear mappings from anchor parameters to task-specific domain knowledge. We show that, for general CNNs, the CLR parameter increase is less than 0.6% for any new task. Our method outperforms 13 state-of-the-art continual learning baselines on a new challenging sequence of 53 image classification datasets. Code and data are in the top link.*

## 1. Introduction

Continual Learning (CL) focuses on the problem of learning from a stream of data, where agents continually extend their acquired knowledge by sequentially learning new tasks or skills, while avoiding forgetting of previous

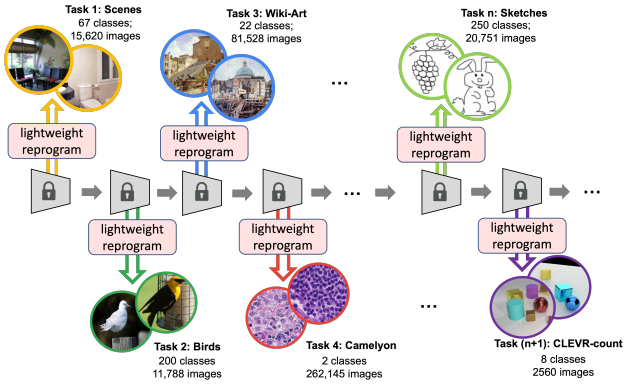


Figure 1. Equipped with a task-agnostic immutable CNN model, our approach "reprogram" the CNN layers to each new task with lightweight task-specific parameters (less than 0.6% of the original model) to learn sequences of disjoint tasks, assuming data from previous tasks is no longer available while learning new tasks.

tasks [40]. In the literature, CL is also referred to as lifelong learning [10, 2, 40] and sequential learning [4]. This differs from standard train-and-deploy approaches, which cannot incrementally learn without catastrophic interference across tasks [21]. *How to avoid catastrophic forgetting* is the main challenge of continual learning, which requires that the performance on previous learned tasks should not degrade significantly over time when new tasks are learned. This is also related to a general problem in neural network design, the stability-plasticity trade-off [23], where plasticity represents the ability to integrate new knowledge, and stability refers to the ability to retain previous knowledge [13].

Dynamic Network methods have been shown to be among the most successful ones to solve continual learning, which usually shows great stability on previous tasks and alleviates the influence of catastrophic forgetting by dynamically modify the network to solve new tasks, usually by network expansion [40, 56, 12, 57]. For stability, a desirable approach would be to fix the backbone and learn extra task-specific parameters on top of it, which will have no catastrophic forgetting. However, the number of parameters in such methods can quickly become very large. How to re-

duce the amount of required extra parameters is still a challenging problem. To solve the above issues, our approach is based on three motivations:

(1) **Reuse instead of re-learn.** Adversarial Reprogramming [16] is a method to "reprogram" an already trained and frozen network from its original task to solve new tasks by perturbing the input space without re-learning the network parameters. It computes a single noise pattern for each new task. This pattern is then added to inputs for the new task and fed through the original network. The original network processes the combined input + noise and generates an output, which is then remapped onto the desired new task output domain. For example, one pattern may be computed to reprogram a network originally trained on ImageNet [46] to now solve MNIST [14]. The same pattern, when added to an image of an MNIST digit, would trigger different outputs from the ImageNet-trained network for the different MNIST classes (e.g., digit 0 + pattern may yield "elephant"; digit 1 + pattern "giraffe", etc). These outputs can then be re-interpreted according to the new task (elephant means digit 0, etc). Although the computation cost is prohibitively high compared to baseline lifelong learning approaches, here we borrow the reprogramming idea; but we conduct more lightweight yet also more powerful reprogramming in the parameter space of the original model, instead of in the input space.

(2) **Channel-wise transformations may link two different kernels.** GhostNet [24] could generate more feature maps from cheap operations applied to existing feature maps, thereby allowing embedded devices with small memory to run effectively larger networks. This approach is motivated by near redundancy in standard networks: after training, several learned features are quite similar. As such, Han *et al.* [24] generate some features as linear transformations of other features. Inspired by this, our approach augments a network with new, linearly transformed feature maps, which can cheaply be tailored to individual new tasks.

(3) **Lightweight parameters could shift model distribution.** BPN [56] adds beneficial perturbation biases in the fully connected layers to shift the network parameter distribution from one task to another, which is helpful to solve continual learning. This is cheaper than fine-tuning all the weights in the network for each task, instead tuning only one bias per neuron. Yet, this approach provided good lifelong learning results. However, the method could only handle fully connected layers and the performance is bounded by the limited ability of the bias parameters to change the network (only 1 scalar bias for each neuron). Our method instead designs more powerful reprogramming patterns (kernels) for the CNN layers, which could lead to better performance on each new task.

Drawing from these three ideas, we propose channel-wise lightweight reprogramming (CLR). We start with task-agnostic immutable parameters of a CNN model pretrained on a relatively diverse dataset (e.g., ImageNet-1k, Pascal VOC, ...) if possible, or on a self-supervised proxy task, which requires no semantic labels. We then adaptively "reprogram" the immutable task-agnostic CNN layers to adapt and solve new tasks by adding lightweight channel-wise linear transformation on each channel of a selected subset of Convolutional layers (Fig 2). The added reprogramming parameters are 3x3 2D convolutional kernels, each working on separate channels of feature maps after the original convolutional layer. CLR is very cheap but still powerful, with the intuition that different kernel filters could be reprogrammed with a task-dependent linear transformation.

The main contributions of this work are:

- We propose a novel continual learning solution for CNNs, which involves reprogramming the CNN layers trained on old tasks to solve new tasks by adding lightweight task-specific reprogramming parameters. This allows a single network to learn potentially unlimited input-to-output mappings, and to switch on the fly between them at runtime.
- Out method achieves better stability-plasticity balance compared to other dynamic network continual learning methods: it does not suffer from catastrophic forgetting problems and requires limited extra parameters during continual learning, which is less than 0.6% of the original parameter size for each new task.
- Our method achieves state-of-the-art performance on task incremental continual learning on a new challenging sequence of 53 image classification datasets.

## 2. Related Work

**Continual learning.** Continual learning methods can be broadly categorized in three approaches [13]: Replay-based, Regularization-based, and Dynamic network-based.

*(1) Replay methods* use a buffer containing sampled training data from previous tasks, as an auxiliary to a new task's training set. The buffer can be used either at the end of the task training (iCaRL [43], ER [44]) or during training (GSS, AGEM, AGEM-R, DER, DERPP [37, 3, 10, 7]). Rehearsal schemes [43, 45, 27, 11] explicitly retrain on a limited subset of stored samples while training on new tasks. [37, 10] use constrained optimization as an alternative solution leaving more leeway for backward/forward transfer. Pseudo rehearsal methods use output of previous model [44] or use generates pseudo-samples with a saved generative model [51]. Rehearsal methods have the drawback that (1) they require an extra buffer to save samples (related to dataset hardness), (2) they easily overfit to the subset of stored samples during replay. (3) the performance seems to be bounded by joint training, which is influenced by the number of tasks.

*(2) Regularization methods* add an auxiliary loss term

to the primary task objective to constraint weight updates. Data-focused methods (LwF[35], LFL[30], DMC [60]) use knowledge distillation from previous models trained on old tasks to constrain the model being trained on the new tasks. Prior-based methods (EWC [31], IMM [34], SI [59], MAS [1]) estimate the importance of network parameters to previously learned tasks, and penalize changing important parameters while training new tasks. The drawbacks of these methods are expensive computation of importance estimation and suboptimality on new tasks, especially with large numbers of tasks.

*(3) Dynamic network methods* dynamically modify the network to solve new tasks, usually by network expansion. When no constraints apply to architecture size, one can grow new branches for new tasks, while freezing previous task parameters [47, 58], or dedicate a model copy to each task [2]. To save memory, recent methods keep architecture remains static, with fixed parts allocated to each task. Previous task parts are masked out during new task training, either imposed at parameters level [18, 38], or unit level [50]. SUPSUP [57]), PSP [12] assign a fixed set of model parameters to a task and avoid over-writing them when new tasks are learned. Dynamic network methods have been shown to be among the most successful ones in solving continual learning. For stability, a desirable approach would be to fix the backbone and learn extra task-specific parameters on top of it, which will have no catastrophic forgetting. SUPSUP [57] uses a randomly initialized fixed backbone and learns a task-specific supermask for each new task, while the performance is bounded by the ability of the fixed backbone. CCLL [52], and EFTs [54] use fixed backbones trained on the first task, and learn task-specific group convolution parameters. The performance is sensitive to the first task, and the extra group convolution hyper-parameter is not straightforward to train.

Our channel-wise lightweight reprogramming method also belongs to dynamic network continual learning methods. Inspired by ghost networks, BPN, and Adversarial reprogramming, we reprogram the immutable task-agnostic parameters and apply lightweight (computation and memory cheap) extra parameters to learn new tasks.

**Meta learning.** Meta learning aims to improve the learning algorithm itself, given the experience of multiple learning episodes [26]. In contrast, conventional ML improves model predictions over multiple data instances. During meta-learning, an outer (or upper/meta) algorithm updates and improves the inner learning algorithm (e.g., one image classification task). For instance, the outer objective could be the generalization performance or learning speed of the inner algorithm. Meta-learning has the following main difference with CL: similar to transfer learning, Meta-learning focuses on finding a better learning strategy or initialization, and cares about the performance on a new or generalized task, while performance on previous tasks is not the interest. Thus, meta-learning could be used as a preliminary step in CLR, to optimize our shared initial parameter $\theta$ in CLR. Meta-learning can learn from a single task in multiple domains [53, 19] or different tasks [20, 36, 5], which requires access to some data from all base tasks or domains at the same time, while our method and CL assume sequential learning and cannot access the data from previous tasks. Also, meta-learning assumes the learned general setting will be used in similar tasks as the base tasks, while CL does not assume that previous tasks are similar to future tasks.

## 3. Proposed Method

**Problem setting.** In this paper, we focus on the task-incremental setting of continual learning, where data arrives sequentially in batches, and one batch corresponds to one task, such as a new set of classes or a new dataset to be learned. For a given continual learning algorithm, the goal is to obtain a high average performance on all previously learned tasks after learning the final task. At test time, just like PSP [12], CCLL[52], EFTs[54], EWC [31], ER[44], etc, we assume that a task oracle is provided during inference, showing which task a given sample belongs to.

**Structure.** We first introduce how our proposed Channel-wise Lightweight Reprogramming parameters reprogram the immutable task-agnostic backbone by conducting channel-wise linear transformation after the original convolutional layer to change the original Conv-block as CLR-Conv block, and then develop a new CLR-reprogrammed network to solve new task. (Sec. 3.1). Then, we introduce how CLR-reprogrammed networks can be used to solve continual learning tasks (Sec. 3.2).

### 3.1. Channel-wise Lightweight Reprogramming

Our proposed Channel-wise Lightweight Reprogramming method is equipped with an immutable task-agnostic backbone and creates task-specific lightweight reprogramming parameters to solve new tasks. Here, we use a fixed backbone as a task-shared immutable structure. This differs from SUPSUP [57], which uses a randomly initialized fixed backbone, and CCLL [52], EFTs [54], which use fixed backbones trained on the first task. We use a more general and compatible way with less requirement to obtain the backbone: the fixed backbone could be pretrained with supervised learning on a relatively diverse dataset (e.g., on ImageNet-1k [46], or Pascal VOC [17]), or with self-supervised learning on proxy tasks, such as DINO [9] and SwAV [8], which requires no semantic labels – we will see that our approach is robust to the choice of a pretraining dataset in the Sec. 4.5 experiments). This fixed backbone can provide a diverse set of visual features. However, those need to be reprogrammed later for individual tasks, which is achieved by our CLR layers.

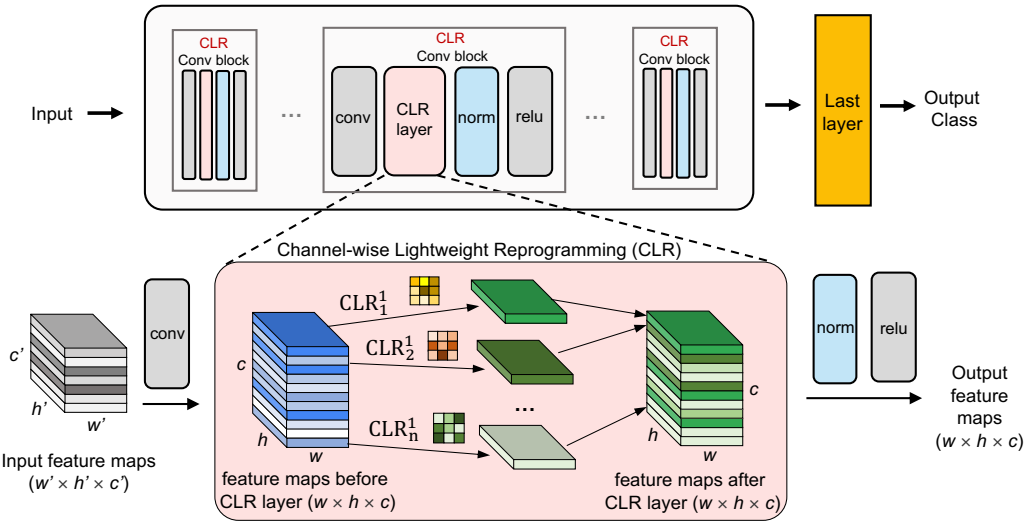


Figure 2. Proposed continual learning model with channel-wise lightweight reprogramming (CLR) layers. All gray blocks are fixed parameters. (top) General network architecture. (bottom) Details of CLR reprogramming layer: for each channel $k \in [1..c]$ of an original $w \times h \times c$ feature map (blue), a 3x3 kernel is learned to reprogram the feature towards the new task (green), without modifying the original conv parameters (grey).

Specifically, we use channel-wise linear transformation to reprogram the feature map generated by an original convolutional kernel, to generate a new task-specific feature map for the new task.

**Algorithm 1** CLR Layers

**Input:** Feature map $X'_{w\times h\times c}$ (output from the original conv layer $F$), $3 \times 3$ 2D $CLR$ reprogram kernels $\times c$
**Output:** Reprogrammed feature map $\hat{X}'_{w\times h\times c}$
$p \leftarrow \lfloor 3/2 \rfloor$
$X' \leftarrow$ zero-padding($X'$, $p$)
**for** $k \in [1..c]$ **do**
$\hat{X}'[k] \leftarrow CLR_k(X'[k])$ $X'[k]$ *is the k-th channel*
**end for**

Fig. 2 shows the structure of the proposed CLR. CLR is compatible with any convolutional neural network (CNN). A CNN usually consists of several Conv blocks (e.g., Residual block) (Fig. 2 top), which contain a convolutional layer (conv), normalization layer (e.g., batch normalization), and activation layer (e.g., relu). Our method treats a pretrained CNN as a general and task-agnostic immutable parameter backbone for all future tasks, so we fix its parameters. (More details on the choice of a pretraining backbone in the Sec. 4.5 experiments). To reprogram the CNN to solve a new task in continual learning, we add lightweight trainable parameters by changing each of the original Conv blocks into a CLR-Conv block, thereby creating a CLR-reprogrammed CNN (Fig. 2 top). Specifically, as shown in Fig. 2 (top), a CLR-Conv block is obtained by adding a channel-wise lightweight reprogramming layer (CLR layer) after each of the original fixed conv layers. (For parameter saving and preventing overfit, $1\times1$ conv layers are excluded.) Each CLR layer conducts linear transformations on each channel of the original feature map after the fixed conv layer to reprogram the features. Here, the linear transformation is represented with 3x3 2D convolutional kernels conducted on single channel feature map. Fig. 2 (bottom) illustrates the details of Channel-wise linear transformation for reprogramming. For each convolutional kernel $f_k()$, given the input feature $X$, we obtain one channel of feature $x'_k = f_k(X)$ after the process, all Channel-wise features form the output feature map $X'$. Our Channel-wise reprogramming Linear transformation is applied on each channel $x'_k$ of the output feature map $X'$. For each kernel $f_k()$, we have a corresponding reprogramming 3x3 2D kernel $CLR_k$, which takes the single channel output $x'_k$ as input and conducts a linear transformation to obtain the reprogrammed feature $\hat{x'}_k$:

$$\hat{x'}_k = CLR_k(x'_k) = CLR_k(f_k(X)) \tag{1}$$

Algorithm 1 shows the pseudocode of the features before and after a CLR layer. We initialize the CLR layer as an identity transformation kernel (e.g., in a 3x3 2D kernel, the center parameter is one and all others are zero). This setting is crucial for training efficiency and performance, as it favors keeping the general feature extractors in the original fixed model, while at the same time it allows achieving adaptive reprogramming, based on the loss function for the new task.

For CLR-reprogrammed CNNs, the original conv layers in the backbone are fixed, the trainable parameters include the CLR layer after each fixed conv layer (normalization layer is optional to train), and the last fully-connected layer. For CLR-reprogrammed Resnet-50 [25], the trainable CLR layer takes only 0.59% of the parameter of the original fixed Resnet-50 backbone. This efficient parameter property makes CLR-reprogrammed networks easy to deploy for continual tasks.

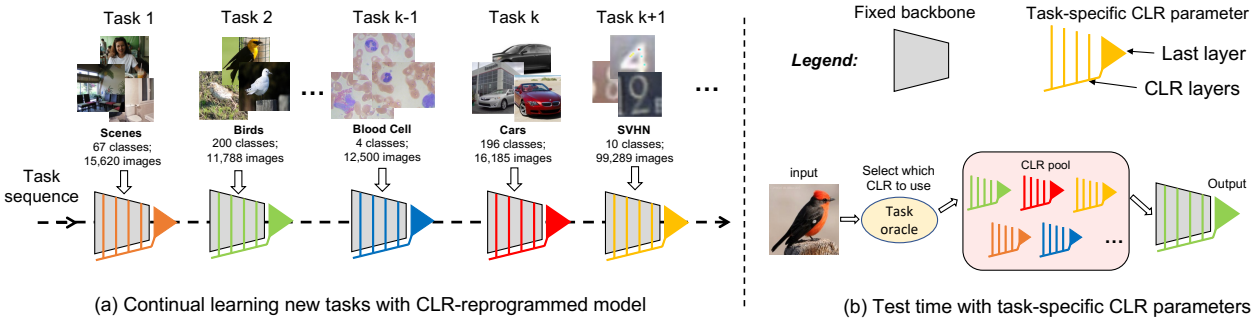


Figure 3. CLR-reprogrammed CNNs for continual learning. (a) In learning time, a CNN model could be reprogrammed by Channel-wise Lightweight Reprogramming parameters to solve continual new tasks. Only the CLR layers need to be trained in each reprogramming. (b) In test time, task oracle will select which task-specific CLR parameters to use and make the final decision.

### 3.2. CLR for Continual Learning

In task incremental continual learning, the model faces a sequence of tasks. As shown in Fig. 3(a), during continual learning, a CLR-reprogrammed model learns each task at a time, and all tasks share the same fixed pretrained backbone. Each task has a trainable task-specific CLR parameter set consisting of the CLR layers after each original conv layer and the last linear layer. During testing (Fig. 3(b)), we assume a perfect task oracle (as assumed by our baselines shown in the next section) which tells the model which task the test image belongs to. The fixed backbone equipped with the corresponding task-specific CLR parameter makes the final decision.

Due to the absolute parameter isolation in CLR (i.e., CLR layers are completely specific and separate for each task, and the shared backbone is not changed at all), our method's performance on every task is not influenced by increasing number of tasks (similar to SUPSUP [57], CCLL [52], and EFT [54]). Theoretically, the CLR-reprogrammed model can learn as many tasks as needed in a continual learning setting and keep the optimal performance on each task with no accuracy decrease when the number of tasks increase, but with a 0.59% increase in parameters for each new task.

## 4. Experiments and results

In this section, we compare our CLR-reprogrammed model to baselines on a challenging 53-dataset with large variance (Sec. 4.1). We evaluate task accuracy fluctuation after learning new tasks to assess forgetting (Sec. 4.2). Then, we evaluate the average accuracy on all learned tasks so far during continual learning (Sec. 4.3). We analyze the network parameters and computation cost during continual learning (Sec. 4.4). We conduct ablation study to analyze the influence of different immutable backbones (Sec. 4.5)

### 4.1. Dataset and Baselines:

**Datasets.** We use image classification as the basic task framework. We extend the conventional benchmark 8-dataset [1, 2] to a more challenging 53-datasets by collecting more challenging classification tasks. 53-datasets consist of 53 image classification datasets. Each one supports one complex classification task, and the corresponding dataset is obtained from previously published sources, e.g., task 1 [41]: classify scenes into 67 classes, such as kitchen, bedroom, gas station, etc (scene dataset with 15,523 images); task 2 [55]: classify 200 types of birds, such as Brewer Blackbird, Cardinal, Chuck will Widow, etc (birds dataset with 11,787 images). A full list and more details on each dataset are in supplementary materials. The 53-datasets is a subset of SKILL-102 [22] Lifelong Learning benchmark dataset[1], more details about dataset creation and an extended version (107 tasks) is in USC-DCT (Diverse Classification Tasks), a broader effort in our laboratory. DCT can be used for many machine vision purposes, not limited to lifelong learning.

We use 53 datasets with $>$ 1.8M images from 1,584 classes over 53 tasks for experiments. Table 1 shows the details of our 53-dataset in comparison to other benchmark datasets. Specifically, we compare the number of different classification targets among different benchmarks, which represents the diversity and difference among different continual tasks. For instance, our 53-dataset contains 5 different classification targets: object recognition, style classification (e.g., painting style), scene classification, counting number (e.g., CLVER dataset [29]), and medical diagnosis (e.g., Breast Ultrasound Dataset). To date, our 53-dataset is one of the most challenging image classification benchmarks for continual learning algorithms, with a large number of tasks and inter-task variance.

For the experiments below, we subsampled the dataset to allow some of the sequential baselines to converge: we capped the number of classes/task to 300 (only affected 1 tasks), and used either around $5,120$ training images for tasks with $c \geq 60$ classes, or around $2,560$ for tasks with $c < 60$, where $c$ represent the number of classes. Thus, we used 53 tasks, total 1,583 classes, total 132,625 training images and 13,863 test images. We also conduct experiments on CIFAR-100 dataset (Appendix Fig.12). **Baselines.** As discussed in Sec 1, we grant each baseline a perfect task or-

[1]SKILL-102 dataset http://ilab.usc.edu/andy/skill102

| Comparison | 53-dataset (ours) | 8-dataset [1, 2] | ImageNet [46] | Fine grained 6 tasks [46] [34] | Cifar100 [33] | F-CelebA [42] |
|---|---|---|---|---|---|---|
| # tasks | **53** | 8 | 20 | 6 | 20 | 10 |
| # Classes | 1,584 | 738 | 1000 | **1943** | 100 | 20 |
| # Images | **1,811,028** | 166,360 | 1,300,000 | 1,440,086 | 60,000 | 1,189 |
| # different classification target | **5** | 1 | 1 | 2 | 1 | 1 |
| Mix image style (nature/artifact) | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ |
| Mix super/fine-class classification | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |

Table 1. Comparison of 53-dataset with other benchmark datasets including Cifar-100 [33], F-CelebA [42], Fine-grained 6 tasks [46] [34], [39], [32], [48], [15]. Note that our 53-dataset covers the 8-dataset, F-CelebA and part of the Fine grained 6 tasks.

acle during inference. We implemented 13 baselines, which can be roughly categorized in the following 3 categories [13]:

*(1) Dynamic Network methods* contains most of the baseline methods because our method also belongs to it: they dynamically modify the network to solve new tasks, usually by network expansion. We use PSP [12], Supermask in Superposition (SUPSUP) [57], CCLL[52], Confit[28], and EFTs[54] as the representative methods of this category: For PSP, the model learns all 53 tasks in sequence, generating a new PSP key for each task. The keys help segregate the tasks within the network in an attempt to minimize interference. For SUPSUP, the model uses a random initialized parameter as fixed backbone and learns class-specific supermasks for each task, which help alleviate catastrophic forgetting. During inference, different tasks use different supermasks to make the decision.

*(2) Regularization methods* add an auxiliary loss term to the primary task objective to constraint weight updates. The extra loss can be a penalty on the parameters (EWC [31], LwF[35], MAS [1] and SI [59]) or on the feature-space (FDR [6]), such as using Knowledge Distillation (DMC [60]). We use EWC, online-EWC, SI, LwF as the representatives of this category: for EWC, one agent learns all 53 tasks in sequence, using EWC machinery to constrain the weights when a new task is learned, to attempt to not destroy performance on previously learned tasks.

*(3) Replay methods* use a buffer containing sampled training data from previous tasks, as an auxiliary to a new task's training set. The buffer can be used either at the end of the task training (iCaRL, ER [43, 44]) or during training (GSS, AGEM, AGEM-R, DER, DERPP [37, 10, 3, 7]). We use Episodic Memory Rehearsal (ER) as the representative baseline of this category: One agent learns all 53 tasks in sequence. After learning each task, it adds 10 images/class of that task to a growing replay buffer that will later be used to rehearse old tasks. When learning a new task, the agent learns from all the data for that task, plus rehearses all old tasks using the memory buffer. Additionally, SGD is a naive baseline that just fine-tunes the entire network for each task, with no attempt at minimizing interference. SGD-LL is a variant that uses a fixed backbone plus a single learnable shared last layer for all tasks with a length equal to the task with the largest number of classes (500 classes in our setting). SGD uses standard stochastic gradient descent as optimization, which may suffer from the catastrophic forgetting problem.

For a fair comparison, all the 13 baseline methods use an ImageNet-pretrained ResNet-50 [25] backbone, except for PSP, Confit (requires ResNet-18) and SUPSUP (which requires a randomly initialized ResNet-50 backbone).

### 4.2. Accuracy on the first tasks

To evaluate the performance of all methods in overcoming catastrophic forgetting, we track the accuracy on each task after learning new tasks. If the method suffers from catastrophic forgetting, then the accuracy of the same task will decrease after learning new tasks. A great continual learning algorithm is expected to maintain the accuracy of the original learning performance after learning new tasks, which means that old tasks should be minimally influenced by an increasing number of new tasks. Fig. 4 shows the accuracy on the first, and second tasks as we learn from 1 to 53 tasks using 53-datasets, to gauge the amount of forgetting on early tasks as many more tasks are learned. Overall, our CLR-reprogrammed model maintains the highest accuracy on these early tasks over time, and importantly, our method (similar to CCLL [52], EFT [54] and SUPSUP [57]) avoids forgetting and maintains the same accuracy as the original training, no matter how many new tasks are learned. SUPSUP is not able, even from the beginning, to learn task 1 as well as other methods. We attribute this to SUPSUP's limited expressivity and capacity to learn using masks over a random backbone, especially for tasks with many classes. Indeed, SUPSUP can perform very well on some other tasks, usually with a smaller number of classes (e.g., SVHN, UMNIST Face Dataset). Baseline methods suffer a different degree of forgetting: EWC [50], PSP [12], ER [44], SI [59], and LwF [35] suffers severe catastrophic forgetting in the challenging dataset. We noticed similar performance on the second task.

### 4.3. Average accuracy after learning all 53 tasks

We computed the average accuracy on all tasks learned so far after learning 1, 2, 3, ... 53 tasks. We plot the accuracy averaged over all tasks learned so far, as a function of the number of tasks learned in Fig. 5. Note that the level of difficulty for each of the 53 tasks is quite variable, as shown in Fig. 6. Hence, the average accuracy over all tasks so far may go up or down when a new task is added, de-

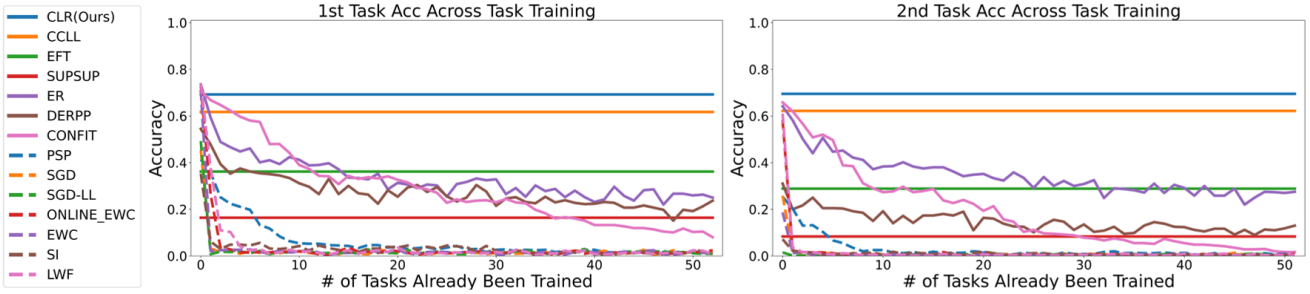


Figure 4. Accuracy on task 1 as a function of the number of tasks learned. Our approach maintains the highest accuracy on task 1 over time, and importantly, it totally avoids catastrophic forgetting and maintains the same accuracy as the original training, no matter how many new tasks are learned. As discussed in the approach section, this is because we explicitly isolate the task-specific parameters for all tasks and avoid parameter interference. This is also the case for baselines SUPSUP [57], CCLL [52], and EFT [54]. Other baseline methods suffer a different degree of catastrophic forgetting. EWC [50], PSP [12], LwF[35], SI [59] and SGD suffer severe catastrophic forgetting with this challenging dataset. Rehearsal-based method ER performs relatively well because it has an unlimited large replay buffer, and it saves 10 images/class of the previous tasks. Yet, the overall accuracy of ER is still lower than our CLR-reprogrammed model. Rehearsal methods also incur higher (and increasing) training costs because of the rehearsing. We noticed similar performance on the second task.

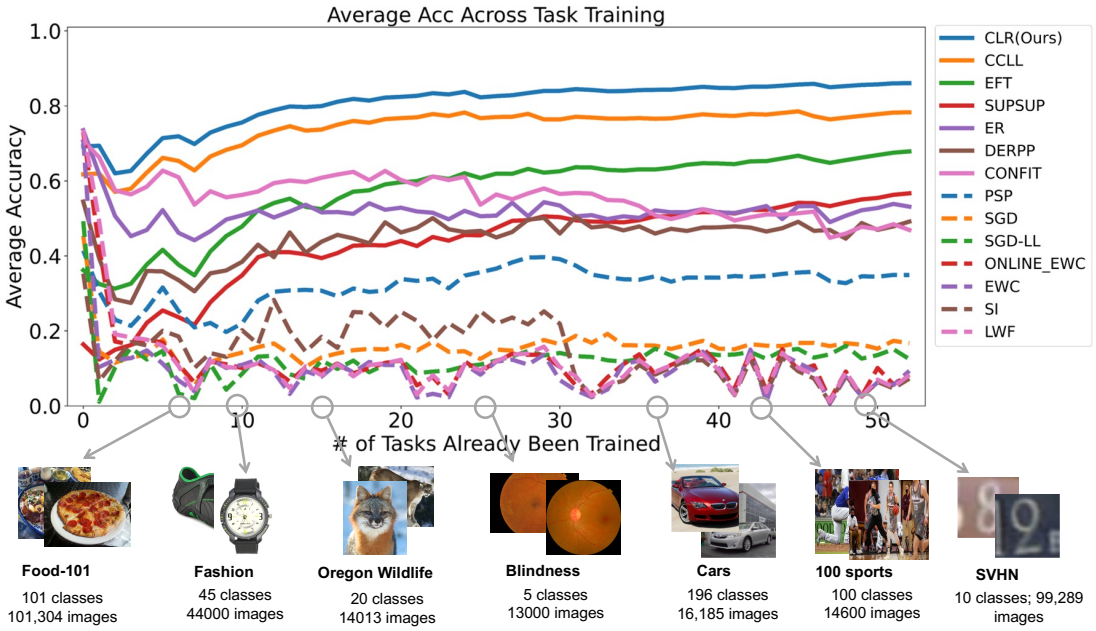


Figure 5. Average accuracy on all tasks learned so far, as a function of the number of tasks learned. Our CLR-reprogrammed approach is able to maintain higher average accuracy than all baselines. The average accuracy increases because some of the later tasks are easier than former tasks (i.e., later tasks have higher accuracy).

pending on whether an easier or harder task is added. The average accuracy represents the overall performance of a continual learning method in learning and then performing sequential tasks. Overall our CLR-reprogrammed model achieves the best average accuracy compared with all baselines. For replay-based methods, the overall performance is lower than us even though they have a large buffer, and the training time is increased for these methods with the increase in task number. EWC, PSP, LWF, SI suffer severe catastrophic forgetting in the challenging 53-datasets.

To show more details, we plot the accuracy of each task after learn all 53 tasks with our CLR-reprogrammed method in Fig. 6. This shows that 53-dataset provides a range of difficulty levels for the various tasks, and is quite hard overall.

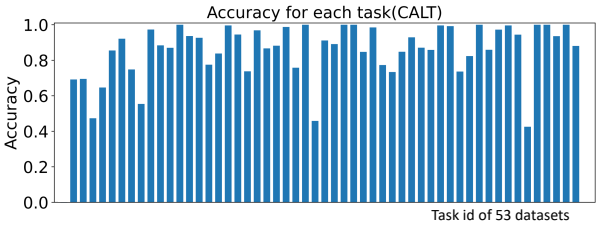


Figure 6. Absolute accuracy per task after learning 53 tasks with our CLR-reprogrammed CNN.

### 4.4. Parameter and computation cost

Achieving higher overall average accuracy during learning of sequential tasks is very important. A great contin-

| Method | Extra parameter to add 1 new task | Computation cost | Average Acc (53-datasets) |
|---|---|---|---|
| SGD | **0%** | 1 | 16.71 % |
| PSP [12] | 5.02% | 0.828 | 34.91 % |
| EWC [31] | 0% | 1.160 | 9.36 % |
| ONLINE EWC [49] | 0% | 1.011 | 9.47* % |
| SGD-LL | **0%** | **0.333** | 12.49 % |
| ER [44] | 189.14% | 3.99 | 53.13 % |
| SI [59] | 0% | 1.680 | 7.28% |
| LwF [35] | 0% | 1.333 | 8.23% |
| SUPSUP [57] | 3.06% | 1.334 | 56.69 % |
| EFT [54] | 3.17% | 1.078 | 67.8 % |
| CCLL [52] | 0.62% | 1.006 | 78.3 % |
| CLR (Ours) | 0.59% | 1.003 | **86.05 %** |

Table 2. Extra parameter expenditures and computation cost analysis. We treat the computation cost of SGD as the unit, and the computation costs of other methods are normalized by the cost of SGD. PSP's low computation cost comes from using a Resnet-18 backbone instead of Resnet-50 which is its original form. For EWC, though the final model size does not increase, the performance is poor and N fisher matrices are needed during training; EWC-online updated the way of updating the fisher matrix and only requires one fisher matrix during training. ER maintain a memory buffer that includes five images per class from the tasks that have already been seen, we spread the size in bytes of the image buffer over the 53 tasks to obtain the amount of extra parameters per task. SUPSUP requires a 3MB mask for each task.

ual learning algorithm is also expected to minimize the requirement of extra network parameters, and the computation cost. Table. 2 shows the required extra parameter and computation cost for our approach and the baselines. The "extra parameters to add one new task" represents the percentage compared with the original backbone (e.g., ResNet-50) size. For instance, EWC needs no extra parameter, while its computation cost is relatively high (to compute the Fisher information matrix that will guide the constraining of parameters while learning a new task), and the accuracy is not the best. Our CLR method required only 0.59% extra parameters to learn each new task, and the computation cost increase is small compared with baseline SGD (normal training). Importantly, our method achieves the best average accuracy.

### 4.5. Influence of different immutable backbone

Our method obtains the task-agnostic immutable parameters by training the CNN model on a relatively diverse dataset with supervised learning or proxy tasks with self-supervised learning. To investigate the influence of different pretraining methods on the performance of continual learning, we choose four different kinds of task-agnostic immutable parameters trained with different datasets and tasks. For supervised learning, besides Imagenet-1K, we also conduct experiments with a pretrained backbone on the Pascal-VOC image classification task (relatively smaller one). For self-supervised learning, which needs no semantic labels, we conduct experiments with backbone trained with DINO [9] and SwAV [8]. DINO [9] is a self-distillation with no label framework, which utilizes multiple crop of the image (patch) on the same model and update model's parameters with exponential moving average. While SwAV [8] simultaneously clusters the data and keeps the consistency between cluster assignments produced for different augmentations of the same image. The results in Table 3 shows that both supervised and self-supervised learning could contribute a good immutable parameters, and our method is robust to different backbone pre-training. Note how Pascal-VOC is a much smaller dataset, which may explain the lower overall accuracy; with any of the other (larger) datasets, accuracy is almost the same, suggesting that our method is not highly dependent on a specific set of backbone features.

| Learning paradigm | Method/dataset | Average Acc (53-datasets) |
|---|---|---|
| Supervised Learning | ImageNet-1k [46] | 86.05 % |
| Supervised Learning | Pascal VOC [17] | 82.49 % |
| Self-supervised Learning | SwAV [8] | 85.12 % |
| Self-supervised Learning | DINO [9] | 85.77 % |

Table 3. Influence of different task-agnostic immutable parameter. Both supervised learning and self-supervised learning could contribute a relatively good immutable parameter for our method, which shows that our method is robust to different backbones.

## 5. Conclusion

We propose Channel-wise Lightweight Reprogramming (CLR), a parameter-efficient add-on continual learning method, to allow a single network to learn potentially unlimited parallel input-to-output mappings and to switch on the fly between them at runtime. CLR adds channel-wise lightweight linear reprogramming to shift the original pre-trained fixed parameter to each task, which is simple and generalizable to any CNN-based model. The experiments on continually learning 53 different and challenging tasks show that the CLR method achieves state-of-the-art performance on task incremental continual learning. Besides high performance, CLR is also parameter-efficient, which requires only 0.59% extra parameter to learn a new task.

**Acknowledgements** This work was supported by Amazon ML Fellowship, C-BRIC (one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA), DARPA (HR00112190134) and the Army Research Office (W911NF2020053). The authors affirm that the views expressed herein are solely their own, and do not represent the views of the United States government or any agency thereof.

## References


[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision*, pages 139–154, 2018. 3, 5, 6

[2] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3366–3375, 2017. 1, 3, 5, 6

[3] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in Neural Information Processing Systems*, 2019. 2, 6

[4] Rahaf Aljundi, Marcus Rohrbach, and Tinne Tuytelaars. Selfless sequential learning. *arXiv preprint arXiv:1806.05421*, 2018. 1

[5] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in Neural Information Processing Systems*, 2016. 3

[6] Ari S Benjamin, David Rolnick, and Konrad Kording. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018. 6

[7] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Advances in Neural Information Processing Systems*, pages 15920–15930, 2020. 2, 6

[8] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in neural information processing systems*, 33:9912–9924, 2020. 3, 8

[9] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021. 3, 8

[10] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018. 1, 2, 6

[11] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019. 2

[12] Brian Cheung, Alex Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno Olshausen. Superposition of many models into one. *arXiv preprint arXiv:1902.05522*, 2019. 1, 3, 6, 7, 8

[13] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, 2021. 1, 2, 6

[14] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. 2

[15] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Transactions on Graphics (TOG)*, 31(4):1–10, 2012. 6

[16] Gamaleldin F Elsayed, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial reprogramming of neural networks. *arXiv preprint arXiv:1806.11146*, 2018. 2

[17] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, Jan. 2015. 3, 8

[18] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017. 3

[19] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135, 2017. 3

[20] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1568–1577. PMLR, 2018. 3

[21] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999. 1

[22] Yunhao Ge, Yuecheng Li, Di Wu, Ao Xu, Adam M. Jones, Amanda Sofie Rios, Iordanis Fostiropoulos, shixian wen, Po-Hsuan Huang, Zachary William Murdock, Gozde Sahin, Shuo Ni, Kiran Lekkala, Sumedh Anand Sontakke, and Laurent Itti. Lightweight learner for shared knowledge lifelong learning. *Transactions on Machine Learning Research*, 2023. 5

[23] Stephen Grossberg. *Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition, and Motor Control*. Springer Science & Business Media, 1982. 1

[24] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1580–1589, 2020. 2

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4, 6

[26] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5149–5169, 2021. 3

[27] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018. 2

[28] Shibo Jie, Zhi-Hong Deng, and Ziheng Li. Alleviating representational shift for continual fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3810–3819, 2022. 6

[29] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2901–2910, 2017. 5

[30] Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. Less-forgetting learning in deep neural networks. *arXiv preprint arXiv:1607.00122*, 2016. 3

[31] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. 3, 6, 8

[32] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013. 6

[33] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009. 6

[34] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. *Advances in Neural Information Processing Systems*, 2017. 3, 6

[35] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017. 3, 6, 7, 8

[36] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 3

[37] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017. 2, 6

[38] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018. 3

[39] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729, 2008. 6

[40] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. 1

[41] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 413–420, 2009. 5

[42] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, 2017. 6

[43] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017. 2, 6

[44] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995. 2, 3, 6, 8

[45] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in Neural Information Processing Systems*, 2019. 2

[46] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge (2014). *arXiv preprint arXiv:1409.0575*, 2(3), 2014. 2, 3, 6, 8

[47] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 3

[48] Babak Saleh and Ahmed Elgammal. Large-scale classification of fine-art paintings: Learning the right metric on the right feature. *arXiv preprint arXiv:1505.00855*, 2015. 6

[49] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pages 4528–4537. PMLR, 2018. 8

[50] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4548–4557, 2018. 3, 6, 7

[51] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Advances in Neural Information Processing Systems*, 2017. 2

[52] Pravendra Singh, Vinay Kumar Verma, Pratik Mazumder, Lawrence Carin, and Piyush Rai. Calibrating cnns for lifelong learning. *Advances in Neural Information Processing Systems*, 33:15579–15590, 2020. 3, 5, 6, 7, 8

[53] Sebastian Thrun and Lorien Y. Pratt. Learning to learn: Introduction and overview. In *Learning to Learn*, 1998. 3

[54] Vinay Kumar Verma, Kevin J Liang, Nikhil Mehta, Piyush Rai, and Lawrence Carin. Efficient feature transformations for discriminative and generative continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13865–13875, 2021. 3, 5, 6, 7, 8

[55] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 5

[56] Shixian Wen, Amanda Rios, Yunhao Ge, and Laurent Itti. Beneficial perturbation network for designing general adaptive artificial intelligence systems. *IEEE Transactions on Neural Networks and Learning Systems*, 2021. 1, 2

[57] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *Advances in Neural Information Processing Systems*, pages 15173–15184, 2020. 1, 3, 5, 6, 7, 8

[58] Ju Xu and Zhanxing Zhu. Reinforced continual learning. *Advances in Neural Information Processing Systems*, 2018. 3

[59] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995, 2017. 3, 6, 7, 8

[60] Junting Zhang, Jie Zhang, Shalini Ghosh, Dawei Li, Serafettin Tasci, Larry Heck, Heming Zhang, and C-C Jay Kuo. Class-incremental learning via deep model consolidation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1131–1140, 2020. 3, 6