Lifelong Compositional Feature Replays Beat Image Replays in Stream Learning

Morgan B. Talbot, Rushikesh Zawar, Rohil Badkundri, Mengmi Zhang and Gabriel Kreiman

* Equal contribution

Address correspondence to mengmi@i2r.a-star.edu.sg; gabriel.kreiman@tch.harvard.edu

1 Harvard-MIT Health Sciences and Technology, Massachusetts Institute of Technology
2 Boston Children’s Hospital, Harvard Medical School
3 Center for Brains, Minds, and Machines
4 Birla Institute of Technology and Science, Pilani
5 Harvard University
6 CFAR and I2R, Agency for Science, Technology and Research

Abstract

Our brains extract durable, generalizable knowledge from transient experiences of the world. Artificial neural networks come nowhere close: when tasked with learning to classify objects by training on non-repeating video frames in temporal order (online stream learning), models that learn well from shuffled datasets catastrophically forget old knowledge upon learning new stimuli. We propose a new continual learning algorithm, Compositional Replay Using Memory Blocks (CRUMB), which mitigates forgetting by replaying feature maps reconstructed by recombining generic parts. Just as crumbs together form a loaf of bread, we concatenate trainable and re-usable “memory block” vectors to compositionally reconstruct feature map tensors in convolutional neural networks. CRUMB stores the indices of memory blocks used to reconstruct new stimuli, enabling replay of specific memories during later tasks. CRUMB’s memory blocks are tuned to enhance replay: a single feature map stored, reconstructed, and replayed by CRUMB mitigates forgetting during video stream learning more effectively than an entire image while occupying only 3.6% of the memory. We stress-tested CRUMB alongside 13 competing methods on 5 challenging datasets, including 3 video stream datasets containing 10, 12, and 14 classes, and 2 image datasets containing 100 and 1000 classes. To address the limited number of existing online stream learning datasets, we introduce 2 new benchmarks by adapting existing datasets for stream learning. With about 4% of the memory and 20% of the runtime, CRUMB mitigates catastrophic forgetting more effectively than the prior state of the art with an average top-1 accuracy margin of 4.4%, achieving the highest accuracy on 6 out of 8 benchmarks. Our code is available at this link.

1. Introduction

Humans adapt to new and changing environments by learning rapidly and continuously. Previously learned skills...
CRUMB, for online stream learning. The composition mechanism is end-to-end trainable and reusable.

- **[New Benchmarks]** We adapted 2 datasets, Toybox [48] and iLab [6], to introduce new online stream learning benchmarks. We tested CRUMB on the new benchmarks plus 3 established continual learning datasets alongside 13 competing methods, showing that CRUMB typically outperforms SOTA by large margins. All source code, results data, and benchmark details are available at [https://tinyurl.com/4zed3ske](https://tinyurl.com/4zed3ske).

- **[Reduced Forgetting]** CRUMB’s trainable codebook captures the essential components needed for reconstructing class-discriminative features, but is also more than the sum of its parts: a reconstructed and replayed feature map reduces forgetting more effectively than an entire raw image. We replicate this surprising result across three online stream learning datasets, with top-1 accuracy improvements between 5.1% and 13.4% (8.8% on average).

- **[Superior Efficiency]** Storing \( n \) compositional feature maps for replay prevents catastrophic forgetting substantially more effectively than storing \( n \) raw images, while only requiring about 3.6% of the memory usage of raw images. Additionally, compared with the next most accurate method (REMIND [14]), CRUMB requires only about 15-22% as much training runtime, and occupies only 3.7% to 4.1% of REMIND’s peak memory footprint.

2. Related work

2.1. Weight regularization

Weight regularization methods typically store weights trained on previous tasks and/or impose constraints on weight updates for new tasks [7, 16, 20, 25, 26, 52]. However, storing the importance of the millions of parameters required by SOTA recognition models across all previous tasks is costly [20, 49]. Moreover, empirical evidence suggests that these weight regularization methods do not mitigate catastrophic forgetting as effectively as architecture adaptation and replay methods [32].

2.2. Architecture adaptation

Architecture adaptation methods expand or re-organize the structure of their neural networks to accommodate new tasks. Approaches include adding groups of new neurons (which does not always scale well) [16, 20, 25, 26, 52], pruning and re-using neurons [13], compressing parameters in a consolidation phase [43], and isolating parts of a larger neural network for each specific task [1, 11, 39, 44]. All of these approaches add significant complexity, and some require explicit labelling of task identities, which is impractical in many online learning applications.
2.3. Image and feature replay

In replay methods, images or features from previous tasks are stored or generated and later shown to the model to prevent forgetting [3, 5, 7, 31, 36, 41, 50]. Replay can be highly effective, but comes with some caveats. Relying on limited sets of replay images can lead to overfitting. Storing a large number of raw images for replay is also highly memory-intensive. To limit memory requirements, generative replay systems complement new tasks with “pseudo-data” that resemble previously encountered data and that are produced by a generative model [4, 27, 29, 42, 45–47]. However, the generative models needed to create adequate synthetic data remain large, memory-intensive, and difficult to train [49].

When memory is limited, REMIND [14] achieves excellent performance in online stream learning by replaying compressed feature maps, allowing it to store many more training items within a fixed memory budget. REMIND compresses feature maps using a product quantizer [18] that must be trained by performing k-means clustering on a large subset of training data stored in memory. This process scales poorly in terms of memory requirements as the size and complexity of training datasets increases. In contrast, CRUMB’s differentiable codebook is trained during classification alongside other network parameters. This leads to 3 advantages over REMIND: (1) Training the codebook with a classification objective, rather than a product quantizer with no objective beyond unsupervised feature clustering, leads to markedly improved stream learning accuracy on most baselines. (2) CRUMB’s codebook is trained in parallel with CNN weights using gradient updates from mini-batches. This improves scalability by dramatically reducing peak memory usage, which spikes during REMIND’s codebook initialization phase as it performs k-means clustering on a large portion of training data (see Table 1). (3) In our implementation, CRUMB’s runtime is about 15-22% of REMIND’s runtime (Table 1).

3. Online stream learning benchmarks

3.1. Training protocols

We consider two incremental class settings for online stream learning protocols [14] (Fig. 1).

Class-instance. Each task contains short video clips of different objects from several classes, and the video clips are presented one after another in random order within each task. An ideal learning algorithm in this setting would be stable enough to remember prior tasks while being sufficiently plastic to learn generalizable class boundaries for new classification tasks, despite encountering many images of each object at once before moving on to the next.

Class-iid. Images/video frames are randomly shuffled within each task but not interspersed among tasks, and are shown only once.

In both settings, our model and all competing baseline models are allowed to train for many epochs on the first task, but are restricted to viewing each image from each task only once in all subsequent tasks. This emulates real-time acquisition of training data that cannot be stored except in a limited-capacity replay buffer.

3.2. Stream learning benchmark datasets

We evaluated our model on three video datasets (class-instance and class-iid protocols), and two image datasets (class-iid only). For all datasets, we used different data/task orderings across training runs. A global holdout test set of images/frame sequences was used for all training runs. To help address the limited number of online video stream datasets, we adapt two datasets originally designed for studying object transformations, Toybox [48] and iLab [6], to the online stream learning setting.

The CORE50 video dataset [30] contains images of 50 objects in 10 classes. Each object has 11 instances, which are 15 second video clips of the object under particular conditions and poses. We followed [14] for the training and testing data split.

The Toybox video dataset [48] contains videos of toy objects from 12 classes. We used a subset of the dataset containing 348 toy objects with 10 instances per object, each containing a different pattern of object motion. We sampled each instance at 1 frame per second resulting in 15 images per instance per object. We chose 3 of the 10 instances for our test set, leaving 7 instances for training.

The iLab (iLab-2M-Light) video dataset [6] contains videos of toy vehicles from 14 classes. We used a subset of the dataset containing 392 vehicles, with 8 instances (backgrounds) per object and 15 images per instance. We chose 2 of the 8 instances for our test set.

CORE50, Toybox, and iLab contain limited numbers of classes. To evaluate our model in long-range online class-incremental learning with many classes, we also include results on the following image datasets.

Online-Imagenet image dataset [10]. We include the standard Online-ImageNet dataset split into 10 tasks with 100 classes each. Only class-iid training is possible because the dataset does not contain videos.

Online-CIFAR100 image dataset [22]. The standard Online-CIFAR100 dataset is similar to Online-Imagenet, but is split into 20 tasks with 5 classes each.

3.3. Baseline algorithms for comparison

All baseline algorithms use a CNN pretrained on ImageNet, and the same training protocols as CRUMB. CRUMB and most baselines use SqueezeNet [17], but due to implementation constraints AAN [28], CoPE [9], GSS
We compared against Elastic Weight Regularization: We compared against Elastic Weight Consolidation (EWC) [20], Synaptic Intelligence (SI) [32], Memory Aware Synapses (MAS) [2], Learning without Forgetting (LwF) [26], and Stable SGD [35].

**Memory Distillation and Replay:** We compared against Gradient Episodic Memory (GEM) [31], Incremental Classifier and Representation Learner (iCARL) [41], Bias Correction (BiC) [50], Gradient Sample Selection (GSS) [3], Continual Prototype Evolution (CoPE) [9], Adaptive Aggregation Network (AAN) [28], REMIND [14], and Rainbow Memory (RM) [5]. For replay methods, we limit the number of examples that can be stored in the buffer to fit within a memory budget that is fixed across all methods.

**Proposed algorithm: CRUMB**

We propose a new continual learning algorithm, Compositional Replay Using Memory Blocks (CRUMB). CRUMB consists of a 2-dimensional convolutional neural network (2D-CNN) augmented by an $n \times d$ codebook matrix $B$. A schematic of CRUMB is shown in Fig. 2, with algorithm details in Alg. 1. CRUMB extracts a feature map from each given image using the early layers of a pre-trained 2D-CNN. CRUMB stores feature maps from a subset of images encountered during training. When CRUMB later encounters a new task, it avoids catastrophic forgetting by replaying feature maps of images from those tasks to the later layers of the network.

To further reduce memory requirements, CRUMB uses its codebook matrix $B$ to reconstruct each feature map: permutations of the rows of $B$ (“memory blocks”) are concatenated into tensors that approximate the original feature maps, and only the indices of activated memory blocks need to be stored. The reconstruction step is differentiable, so the matrix $B$ learns during training to best represent features from diverse classes.

**4.1. Feature extraction and classification**

CRUMB’s CNN backbone is split into two nested functions. The early layers of the network comprise $F(\cdot)$, a “feature extractor,” while the remaining, later layers comprise $P(\cdot)$, a classifier. Since early convolutional layers of CNNs are highly transferable [51], the parameters of $F(\cdot)$ are pretrained for image classification using ImageNet [10] and then fixed during stream learning. CRUMB passes each training image through feature extractor $F(\cdot)$ to obtain feature map $Z$, of size $s \times w \times h$ (number of features, width, height). $Z$ is reconstructed using $B$ to form $\tilde{Z}$, and a class prediction output can then be obtained as $P(\tilde{Z})$.

**4.2. Reconstructing feature maps from memory**

CRUMB produces reconstructed feature map $\tilde{Z}$ using only $Z$ and the contents of its $n \times d$ codebook matrix $B$, where each of the $n$ rows $B_k$ is a “memory block” vector. $Z$ is first partitioned evenly along its feature dimension into $s/d$ tensors, with each tensor $D_f$ of size $d \times w \times h$. Each tensor $D_f$ is further split by spatial location into $w \times h$ vectors, denoted $Z_{f,x,y} \in \mathbb{R}^d$, where $d$ is also the length of each row $B_k$ in the matrix $B$. For each vector $Z_{f,x,y}$ in $Z$, a similarity score $\gamma$ is calculated between it and each memory block $B_k$ as follows:

$$\gamma_{f,x,y,k} = \langle Z_{f,x,y}, \frac{B_k}{\|B_k\|_2} \rangle$$

where $\langle u, v \rangle$ is the dot product of $u$ and $v$, and $\|v\|_2$ is the L2-norm of $v$. Because $B_k$ is normalized, $\gamma_{f,x,y,k}$ is highest for the memory block most similar in vector direction to the given $Z_{f,x,y}$. The memory block $B_k$ with the highest $\gamma$ similarity value replaces $Z_{f,x,y}$ at its corresponding location in $\tilde{Z}$ as follows:

$$\tilde{Z}_{f,x,y} \leftarrow B_{k_{f,x,y}}$$

where $k_{f,x,y} = \text{argmax}_k(\gamma_{f,x,y,k})$

Because $\tilde{Z}$ is reconstructed entirely from memory blocks $B_k$, we can save all information needed to reconstruct $\tilde{Z}$ again later by storing both $B$ and the values of $k$ at each $f, x, y$ location in $\tilde{Z}$. Thus, the feature map for the $i$th training image can be stored as:

$$m_i = (k_{1,1,1}, \ldots, k_{f,x,y}, \ldots, k_{d,w,h})$$

For example, in our main implementation, $Z$ is a $512 \times 13 \times 13$ tensor. $d = 16$ so that $Z$ is split into $32 \cdot 13 \cdot 13 = 5408$ vectors of length 16, which are each replaced in $\tilde{Z}$ by a 16-dimensional memory block from a $256 \times 16$ matrix $B$.

**4.3. Training**

During training, both $Z$ and $\tilde{Z}$ are passed separately through the classifier $P(\cdot)$ to obtain two classification probability vectors $P = P(Z)$ and $\tilde{P} = P(\tilde{Z})$, where the length of $p_i$ and $\tilde{p}_i$ is equal to the total number of classes $C_t$ that have been seen by the time of the current task $t$. The loss function $L$ used for training is a weighted sum of the cross-entropy losses $L_{CE}$ derived from $p$ and $\tilde{p}$. With $y_c$ defined as the ground truth class label of a given image:

$$L(p, \tilde{p}, y_c) = \alpha L_{CE}(p, y_c) + \beta L_{CE}(\tilde{p}, y_c)$$

Larger values of $\alpha$ penalize “direct” prediction errors from $P(Z)$, while larger values of $\beta$ penalize “codebook-out” prediction errors from $P(\tilde{Z})$. Although
our model generates class predictions based on both \( \tilde{Z} \) and \( Z \), we use the empirically more accurate predictions from \( Z \) during inference on the test set. Empirically, the best performance was achieved by including both direct and codebook-out predictions in the loss function for pretraining (\( \alpha = \beta = 1 \)), and then removing the direct loss for stream learning (\( \alpha = 0, \beta = 1 \)) (see Sec. 5.4.3) for analysis. \( \alpha = 0 \) makes the loss function for new batches of images more similar to that used for replay, where only the reconstructed \( \tilde{Z} \) is available. Replacement of \( Z \) by the reconstructed \( \tilde{Z} \) can be viewed as both a method to mitigate catastrophic forgetting and a regularization technique to prevent overfitting. Similar to dropout [23], our model’s reconstruction is applied during training but not at test time.

Importantly, although values in CRUMB’s memory blocks play the role of activation values in their reconstruction of \( \tilde{Z} \), they are trainable parameters of the network. Backpropagation from \( \tilde{Z} \)-based predictions generates gradients for the values in each memory block used for reconstruction, and stochastic gradient descent modifies the memory blocks to improve their ability to facilitate class discrimination.

### 4.4. Initializing the codebook matrix

CRUMB’s performance benefits from targeted initialization and pretraining of its codebook matrix, especially in the class-instance setting. The values in the codebook matrix directly replace those in “natural” feature maps derived from images during training - accordingly, the matrix is initialized using a distribution designed to match that of natural feature maps from a pretrained network.

Stream learning performance was substantially improved by pretraining CRUMB on ImageNet [10] classification with 1000 classes, as compared to applying CRUMB with naive memory blocks to a CNN pretrained on the same task. This allowed the memory blocks to learn useful representations of features from a diverse set of 1000 classes (see Table 2).

### 4.5. Replay to mitigate catastrophic forgetting

In online stream learning (see Sec. 3.1), the model is presented with images \( I_t \) from new classes \( e^{\text{new}} \) in task \( t \) where \( e^{\text{new}} \) belongs to the complement set of
Replay of examples from previous tasks is a proven strategy to mitigate catastrophic forgetting in incremental class settings [3, 7, 41, 50], and feature-level replay can be considerably more memory-efficient than storing raw images [14]. We store compressed representations of feature maps from images in each task, and then replay a batch of stored feature maps after each batch of new images during later tasks to mitigate forgetting.

Some algorithms select representative image examples to store and replay based on different scoring functions [8, 21, 37]. However, random sampling uniformly across classes yields outstanding performance in continual learning tasks [49]. Hence, we adopt a random sampling strategy and store up to \( n_X \) pairs of labels and tensors \((y_i, m_i)\), corresponding to images from old classes \( e^{old} \) of previous tasks. Depending on the number of seen classes \( C_1, ..., C_{t-1} \), the storage for each old class contains \( n_X/C_t \) pairs. \( n_X \) is chosen for each dataset depending roughly on the total number of classes.

5. Results

5.1. Stream learning on video datasets

A naive CNN trained on stream learning benchmarks learns each task effectively, but rapidly and catastrophically forgets all prior tasks in doing so. In contrast, a brute-force approach to overcoming catastrophic forgetting that achieves excellent performance in a stream learning setting is to store all encountered images and corresponding class labels, shuffle them, and exhaustively retrain on the resulting dataset in an offline, iid fashion. This renders the benchmark equivalent to offline incremental class learning [38] (“Upper bound” in Fig. 3). By storing a subset of old examples and using a compositional strategy to both enhance and compress these examples, CRUMB allows CNNs to approach the performance of a brute-force approach with roughly an order of magnitude reduction in training time and a tiny 0.013% fraction (on CORe50) of the memory footprint. Accordingly, given a fixed memory budget, CRUMB outperforms all competing models in all three tested video stream learning datasets in the class-instance setting, often by large margins (Fig. 3). For example, CRUMB’s top-1 accuracy on all tasks after class-instance stream learning exceeds that of iCARL by 53%, 44%, and 58%, GEM by 68%, 56%, and 61%, and REMIND by 0.5%, 4.4%, and 25.7% on CORe50, Toybox, and iLab respectively. CRUMB also approaches the offline upper bound to within 5.4%, 20.3%, and 17.5% on the same datasets, demonstrating that it is highly effective at mitigating catastrophic forgetting.

The less challenging class-iid setting is similar to class-instance in that tasks are learned sequentially without revisiting previous tasks, and that each image is seen by the model only once; however, all images within each task are shuffled in an iid manner. This removes the local temporal correlations introduced by sequential frames in video clips. As with class-instance, CRUMB achieves excellent class-iid learning performance: CRUMB’s top-1 accuracy on all tasks after class-iid learning exceeds that of iCARL by 53%, 50%, and 55%, and of GEM by 68%, 60%, and 66% on CORe50, Toybox, and iLab respectively. CRUMB approaches the offline upper bound to within 3.1%, 16.0%, and 12.8% on the same datasets. The performance of REMIND and CRUMB was comparable on class-iid, with CRUMB’s accuracy 4.5% higher than REMIND’s on CORe50, but REMIND’s accuracy higher by 8.1% and 2.4% on Toybox and iLab respectively.

Overall, the regularization baselines performed poorly in stream learning. This is perhaps due to limited exposure to each task given that each image may be visited only once, or because of overfitting to temporally correlated data, especially in the class-instance setting. Because we used a fixed memory budget for replay methods, CRUMB is able to store many more examples than replay methods based on raw images, such as iCARL [41] and Gradient Episodic Memory [31], leading to reduced forgetting.

5.2. Stream learning on natural image datasets

On Online-CIFAR100, CRUMB’s mean top-1 accuracy after class-iid stream learning exceeds that of REMIND by 8.0%, iCARL by 30%, and GEM by 43%, performing within 23% of the offline upper bound. On Online-Imagenet, CRUMB outperforms REMIND by 2.9%, iCARL by 31%, and GEM by 46%, performing within 7.0% of the offline upper bound (see Fig. 3). The performance of many other competing algorithms that are substantially outperformed by CRUMB and REMIND is detailed in supplementary Table S1.

5.3. Memory and runtime efficiency

The closest competitor to CRUMB in this study in terms of top-1 accuracy is REMIND [14]. Both models require specific pretraining procedures: REMIND’s entails training a product quantizer using k-means clustering of feature vectors, which requires a large portion of training data to be held in memory simultaneously at very high memory cost for large datasets. In contrast, CRUMB’s codebook matrix is trained by backpropagation in tandem with regular CNN parameter updates. This approach requires only 3.4% of the peak RAM usage of REMIND for large datasets such as Online-CIFAR100 and Online-Imagenet. Our implementation of CRUMB also has a runtime only 15-22% as long as REMIND’s (see Table 1).
Figure 3. Top-1 accuracy in online stream learning on video datasets (a) CORe50 (b) Toybox and (c) iLab with sample images in (d) (all class-instance setting), as well as image datasets (e) Online-CIFAR100 and (f) Online-ImageNet (all class-iid setting). All models train on the first task for many epochs, but view each image only once on all subsequent tasks. Accuracy estimates are the mean of results from 10 runs, where each run has different class and image/video clip orderings. Error bars show the root-mean-square error (RMSE) among runs. Class-iid results for CORe50, Toybox, and iLab are in supplementary Fig. S1, and results for all baselines on class-instance and class-iid are in supplementary Table S1.

| Dataset      | Memory (GB) | Runtime (hours) |
|--------------|-------------|-----------------|
|              | Ours | REMIND | Ours | REMIND |
| CIFAR100     | 0.036 | 0.87 | 0.29 | 1.91 |
| Imagenet     | 1.66  | 44.34 | 7.86 | 35.64 |

Table 1. Peak memory usage and runtime comparison. CRUMB uses only 3-4% of REMIND’s peak RAM usage, and its runtime is 75-78% less than REMIND’s.

5.4. Model analysis

To elucidate the importance of CRUMB’s various components, we performed a series of ablation studies and experiments with altered training procedures. For each experiment, both class-iid and class-instance results on CORe50 are included in Table 2, but throughout the text of the model analysis section, we analyze class-instance results except where otherwise stated. Experiment names are in bold throughout this section.

5.4.1 Replay: \( n \) CRUMB feature maps beats \( n \) images

Feature-level replay is the mechanism by which CRUMB prevents catastrophic forgetting. Completely removing replay dramatically reduces accuracy by 61.2%. However, CRUMB does not require a large replay buffer of stored feature maps to mitigate forgetting: reducing the buffer size \( n_X \) (number of images for which feature maps are stored) from 200 (Ours) to 100 (Half capacity) and to 50 (Quarter capacity) had a relatively modest impact with 5.1% and 14.9% top-1 accuracy drops respectively.

The quality of stored replay examples is also important. Ours, which stores memory block indices to compositionally reconstruct up to \( n_X \) feature maps, had 9.8% higher accuracy than storing the same number \( n_X \) of entire raw images (Image replay), even though CRUMB’s reconstruction of feature maps inevitably discards information and uses only 3.6% as much memory. We observe this effect for all three of CORe50, iLab, and Toybox in both class-instance and class-iid settings, with top-1 accuracy gains for Ours over Image replay between 5.1% and 13.4% (8.8% on average). Image replay had 1.3% and 19.0% higher accuracy than Ours on Online-CIFAR100 and Online-ImageNet respectively, but Ours uses only 3.6% as much computer memory.

Replaying high-level features with pretrained memory blocks contributed to CRUMB’s high performance. Storing \( n_X \) low-level feature maps from layer 3 instead of layer
12 (Early feature replay vs Ours) reduced performance by 9.7%, and using non-pretrained memory blocks reduced performance by 8.4%. Pretraining with CIFAR100 (100 classes) instead of ImageNet (1000 classes) decreases accuracy by 10.4%. In Freeze memory, no updates to memory blocks were allowed after pretraining. This did not affect accuracy, indicating that fine-tuning the memory blocks was unnecessary for stream learning on CORE50.

5.4.2 CRUMB can learn with very few memory blocks

CRUMB’s performance did not change dramatically with changes to the number of memory blocks. Reducing the from 256 blocks to 128, 64, 32, or 16 blocks, which effectively shrinks the library of feature combinations available to reconstruct feature maps, did not significantly decrease accuracy - reducing to 8 blocks decreased accuracy by 4%, and reducing to 4 or 2 blocks decreased accuracy by about 12.4%. Increasing to 512 blocks did not significantly increase accuracy. This suggests a saturation effect, where a small number of memory blocks is sufficient to reconstruct a wide variety of feature maps.

### Table 2

| Category               | Experiment name          | Class-instance % avg. accuracy | Class-id % avg. accuracy |
|------------------------|--------------------------|-------------------------------|--------------------------|
| Unablated              | Ours                     | 76.80                         | 79.83                     |
|                        | Early feature replay     | 67.08*                        | 66.76*                   |
|                        | Image replay             | 67.02*                        | 72.60*                   |
|                        | CIFAR100 pretrain        | 66.38*                        | 77.06*                   |
|                        | Random CRUMB             | 68.39*                        | 73.56*                   |
|                        | Freeze memory            | 76.76                         | 79.80                     |
| Replay format          |                          |                               |                           |
|                        | Half capacity            | 71.75*                        | 77.61*                   |
|                        | Quarter capacity         | 61.95*                        | 69.21*                   |
|                        | No replay                | 15.62*                        | 11.70*                   |
|                        | Ours - direct loss       | 71.63*                        | 77.16*                   |
|                        | Ours + direct loss       | 61.22*                        | 62.26*                   |
| Loss functions         | Direct loss              | 53.53*                        | 52.50*                   |
|                        | 1 block                  | 9.56*                         | 9.47*                    |
|                        | 2 blocks                 | 64.50*                        | 72.44*                   |
|                        | 4 blocks                 | 64.28*                        | 76.32*                   |
|                        | 8 blocks                 | 72.72*                        | 80.08                    |
|                        | 16 blocks                | 76.09                         | 78.46*                   |
|                        | 256 blocks (Ours)        | 76.80                         | 79.83                     |
|                        | 512 blocks               | 77.41                         | 79.65                     |

Table 2. Top-1 accuracy on all tasks after completion of stream learning (averages of 5 runs on CORE50) for model analysis experiments. * denotes significant difference from Ours ($p < 0.01$, paired-samples t-tests on batches of 100 images).

5.4.3 Loss from reconstructed features is sufficient

CRUMB’s performance is affected by the choice of components in its loss function. The loss function (Eq. 4) is the weighted sum of two terms, “direct loss” and “codebook-out loss.” Our experiments show that the best performance is achieved when both direct loss and codebook-out loss are included in pretraining, but only codebook-out loss is included during stream learning. Removing direct loss from pretraining (“Ours - direct loss”) results in a 5.2% drop in accuracy in the later stream learning tasks - learning from only reconstructed feature maps from start to finish is sufficient for decent performance. Including only codebook-out loss (“Ours”) in stream learning yields a dramatic 25.1% gain in accuracy compared to using only direct loss (“Direct loss”), and a gain of 15.6% compared to using a weighted sum of direct loss and codebook-out loss (“Ours + direct loss”), despite the fact that only the direct, non-reconstructed feature map is used for inference on the test set.

6. Discussion and conclusion

We developed a novel compositional replay strategy to tackle the problem of online stream learning, in which algorithms must learn tasks incrementally from non-repeating, temporally correlated inputs. Our algorithm, CRUMB, learns a set of “memory blocks” that are selected via content similarity and concatenated to reconstruct feature maps. The indices of selected memory blocks are stored for a subset of training images, enabling memory-efficient replay of feature maps to mitigate catastrophic forgetting. CRUMB achieves SOTA stream learning accuracy across five datasets. Furthermore, CRUMB outperforms replay of an equal number of raw images in online video stream learning by 8.8% top-1 accuracy on average, despite using only 3.6% as much memory as image replay. We do not see a similar performance boost when using non-pretrained memory blocks: pretraining primes CRUMB’s memory blocks to enhance replay of training examples beyond their original pixel-level content.

In addition to pretraining, our model analysis experiments (Table 2) show that the design of CRUMB’s loss function is important. When training on new images, using only “codebook-out loss” from classification on reconstructed feature maps leads to much less forgetting than using “direct loss” from raw feature maps. Only codebook-out loss is available when replaying feature maps reconstructed from the memory buffer: using the same loss function on new images keeps the domain more consistent for the post-reconstruction layers of the network.

Our model analysis also shows that feature maps from a wide variety of images can be reconstructed by concatenating vectors (memory blocks) drawn from a surprisingly small codebook. CRUMB attains its best accuracy with a codebook of as few as 16 memory blocks on CORE50, and still performs well with only 2. Although CRUMB is already highly memory-efficient as the memory blocks themselves occupy negligible space, reducing the number of memory blocks may enable further CPU memory usage optimizations (e.g., 4-bit integers as indices for 16
CRUMB’s superior memory and runtime efficiency makes it ideally suited for settings with limited computational resources. Potential applications include edge computing in mobile devices, and autonomous robots that learn continuously from otherwise unmanageable amounts of incoming sensor data while they explore their surroundings. CRUMB is implemented here for CNNs, but could be applied across different architectures in the future: we are exploring CRUMB for replay in transformer models and for experience replay in continual reinforcement learning. Updating CRUMB’s memory blocks using backpropagation in tandem with network weights is highly effective and efficient, and also raises the possibility of tuning memory blocks for shifting domains on the fly. Although updates to the memory blocks beyond pretraining do not appear important for stream learning on CORe50 (see Sec. 5.4.1), fine-tuning may become necessary in tasks with substantial non-stationarity. Future studies could apply CRUMB to stream learning or reinforcement learning tasks with shifting domains, emulating humans or robots in continuously changing environments.

References

[1] Tameem Adel, Han Zhao, and Richard E Turner. Continual learning with adaptive weights (claw). arXiv preprint arXiv:1911.09514, 2019. 2
[2] 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 (ECCV), pages 139–154, 2018. 4
[3] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. arXiv preprint arXiv:1903.08671, 2019. 3, 4, 6
[4] Craig Atkinson, Brendan McCane, Lech Szymanski, and Anthony Robins. Pseudo-recursal: Solving the catastrophic forgetting problem in deep neural networks. arXiv preprint arXiv:1802.03875, 2018. 3
[5] Jihwan Bang, Heesu Kim, YoungJoon Yoo, Jung-Woo Ha, and Jonghyun Choi. Rainbow memory: Continual learning with a memory of diverse samples. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8218–8227, 2021. 3, 4
[6] Ali Borji, Saeed Izadi, and Laurent Itti. ilab-20m: A large-scale controlled object dataset to investigate deep learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2221–2230, 2016. 2, 3, 12
[7] Arslan Chaudhry, Marc’ Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. arXiv preprint arXiv:1812.00420, 2018. 2, 3, 6
[8] Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. arXiv preprint arXiv:1203.3472, 2012. 6
[9] Matthias De Lange and Tinne Tuytelaars. Continual prototype evolution: Learning online from non-stationary data streams. arXiv preprint arXiv:2009.00919, 2020. 3, 4
[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale @articlemai202online, title=Online continual learning in image classification: An empirical survey, author=Mai, Zheda and Li, Ruiwen and Jeong, Jiwan and Quispes, David and Kim, Huyunwoo and Sanner, Scott, journal=Neurocomputing, volume=469, pages=28–51, year=2022, publisher=Elsevier hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009. 3, 4, 5, 12
[11] 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. 2
[12] Robert M French. Catastrophic forgetting in connectionist networks. Trends in cognitive sciences, 3(4):128–135, 1999. 2
[13] Siavash Golkar, Michael Kagan, and Kyunghyun Cho. Continual learning via neural pruning. arXiv preprint arXiv:1903.04476, 2019. 2
[14] Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. In European Conference on Computer Vision, pages 466–483. Springer, 2020. 2, 3, 4, 6, 15
[15] 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
[16] Xu He and Herbert Jaeger. Overcoming catastrophic interference using conceptor-aided backpropagation. 2018. 2
[17] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. arXiv preprint arXiv:1602.07360, 2016. 3, 15
[18] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. IEEE transactions on pattern analysis and machine intelligence, 33(1):117–128, 2010. 3
[19] Ronald Kemker, Marc McClure, Angelina Ahtinio, Tyler L Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In Thirty-second AAAI conference on artificial intelligence, 2018. 2
[20] 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. 2, 4, 15
[21] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In Proceedings of the
Davide Maltoni and Vincenzo Lomonaco. Continuous learning multiple layers of features from tiny images. 2009. 3, 12

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012. 5

Zheda Mai, Ruiwen Li, Jihwan Jeong, David Quispe, David Lopez-Paz et al. Gradient episodic memory for continual learning. The Lancet Digital Health, 2(6):e279–e281, 2020. 2

Cecilia S Lee and Aaron Y Lee. Clinical applications of continual machine learning. Computer Vision and Image Understanding, 178:102858, 2019. 3

Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In Advances in neural information processing systems, pages 4652–4662, 2017. 2

Zhizhong Li and Derek Hoiem. Learning without forgetting. IEEE transactions on pattern analysis and machine intelligence, 40(12):2935–2947, 2017. 2, 4

Xiaohan Wang, Tengyu Ma, James Ainooson, Seunghwan Gido M van de Ven, Zhe Li, and Andreas S Tolias. Gdumb: A simple approach that questions our progress in continual learning. arXiv preprint arXiv:1805.06370, 2018. 3

Jathushan Rajasegaran, Munawar Hayat, Salman H Khan, Fahad Shahbaz Khan, and Ling Shao. Random path selection for continual learning. Advances in Neural Information Processing Systems, 32, 2019. 2

Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. Psychological review, 97(2):285, 1990. 2

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. 3, 4, 6, 15

Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. Connection Science, 7(2):123–146, 1995. 3

Jonathan Schwarz, Jelena Luketina, Wojciech M Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. arXiv preprint arXiv:1805.06370, 2018. 2

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. PMLR, 2018. 2

Gehui Shen, Song Zhang, Xiang Chen, and Zhi-Hong Deng. Generative feature replay with orthogonal weight modification for continual learning. In 2021 International Joint Conference on Neural Networks (IJCNN), pages 1–8. IEEE, 2021. 3

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In Advances in Neural Information Processing Systems, pages 2990–2999, 2017. 3

Gido M van de Ven, Zhe Li, and Andreas S Tolias. Class-incremental learning with generative classifiers. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 3611–3620, 2021. 3

Xiaohan Wang, Tengyu Ma, James Ainooson, Seunghwan Cha, Xiaotian Wang, Azhar Molla, and Matthiée Kunda. The toybox dataset of egocentric visual object transformations. arXiv preprint arXiv:1806.06034, 2018. 2, 3, 12

Junfeng Wen, Yanshui Cao, and Ruitong Huang. Few-shot self reminder to overcome catastrophic forgetting. arXiv preprint arXiv:1812.00543, 2018. 2, 3, 6
[50] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 374–382, 2019. 3, 4, 6

[51] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? arXiv preprint arXiv:1411.1792, 2014. 4

[52] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 3987–3995. JMLR. org, 2017. 2, 4
Supplementary Material: Lifelong Compositional Feature Replays Beat Image Replays in Stream Learning

S1. Additional CRUMB vs baseline results

We report top-1 accuracy results (measured on all tasks/classes in each dataset at the end of stream learning training) for CRUMB and all competing baseline algorithms on three video streaming datasets (CORe50 [30], Toybox [48], iLab [6]) and two image datasets (Online-CIFAR100 [22], Online-Imagenet [10]) in both class-iid and class-instance training protocols in Table S1. For CRUMB and a subset of baseline algorithms, we illustrate task-by-task top-1 accuracy (on all previously seen classes) for the three video datasets in the class-iid protocol in Fig. S1. Similar class-instance accuracy plots for video datasets, and class-iid plots for image datasets, are shown in Fig. 3 in the main paper.

S2. Additional Model Analysis

This section extends Sec. 5.4 from the main paper to examine the importance of additional components of CRUMB. As in the main paper, we report the results from a series of experiments that compare CRUMB (Ours) to versions of CRUMB with altered components or training procedures, on the CORe50 dataset in both class-instance and class-iid protocols. For comparisons of Ours against Image replay (see Sec. 5.4.1 in main paper), we include results for all five of the datasets in our study: CORe50, Toybox, iLab, Online-CIFAR100, and Online-ImageNet. For all model analysis experiments here and in the main paper, (excluding visualizations in Sec. S2.4), CRUMB and each perturbed version of CRUMB completed five independent runs with different data orderings, comprising five independent runs of ImageNet pretraining followed by one run each of CORe50 stream learning for each of class-instance and class-iid protocols. The results for Ours in model analysis experiments are not identical to those used in our main results for comparison with other models, for two reasons: (1) in our main results, we tuned our model to use slightly different hyperparameters, and (2) in our main results we used a single pretraining run as the basis for 10 stream learning runs. Top-1 accuracy results for supplementary model analysis experiments are summarized in Table S2. Experiment names are in bold throughout this section.

S2.1. CRUMB outperforms raw image replay, with a fixed buffer size, on all tested video datasets

In the main paper (Sec. 5.4.1) we report that a version of CRUMB that is allowed to store memory block indices for at most \( n \) feature maps (Ours) achieves higher stream learning accuracy on CORe50 than an otherwise identical version (Image replay) that is allowed to store and replay \( n \) raw images instead (at much higher memory cost: memory block indices use only 3.6% of the memory footprint of raw images). We replicate this result here for video datasets Toybox and iLab, demonstrating that CRUMB is able to consistently outperform raw-image replay with \( n \) stored examples in video stream learning. Ours attains accuracy 9.8% and 7.2% higher than Image replay on CORe50 in class-instance and class-iid respectively, 5.9% and 5.1% higher on Toybox, and 8.1% and 13.4% higher accuracy on iLab. This result does not hold for the image datasets, however: while using 3.6% as much memory, Ours attained accuracy 1.3% and 19.0% lower than Image replay on Online-CIFAR100 and Online-ImageNet respectively.

S2.2. CRUMB is robust to different memory block sizes, block size affects memory efficiency

CRUMB performs well with a range of memory block sizes. Decreasing the number of elements in each memory block from 8 to 4 (4-dim. blocks) results in a modest decrease in performance, 4.7% and 2.1% on class-instance and class-iid respectively. Increasing the number of elements from 8 to 16 or 32 (16-dim. blocks, 32-dim. blocks), which arguably makes accurate reconstruction of feature maps more challenging because higher-dimensional vectors must be replaced by discrete choices of memory blocks, had negligible impact on performance (see Table S2).

The maximum number of examples stored in CRUMB’s replay buffer (\( n \)) was held constant for the memory block
Table S1. Top-1 accuracy results for all tested models on five datasets, measured as mean top-1 accuracy on all tasks/classes after the completion of stream learning. All values are averaged across 10 independent runs, except for those in the Online-ImageNet column, which are averaged across 5 independent runs. The highest accuracy in each column excluding the offline upper bound (corresponding to the best-performing algorithm in online stream learning) is in bold. Class-instance and class-iid training protocols are abbreviated as “c-inst” and “c-iid” respectively. Algorithm name abbreviations can be found in Sec. 3.3 in the main paper. Class-instance is only applicable to video streaming datasets CORe50, Toybox, and iLab because it involves presenting video frames in temporal order (see Sec. 3.1 in main paper). Due to resource constraints, for Online-ImageNet, we tested a subset of algorithms that showed relatively high performance on other benchmarks.

| Dataset         | Category       | Experiment name | Class-instance | Class-iid |
|-----------------|----------------|-----------------|----------------|-----------|
| CORe50          | Unablated      | Ours            | 76.80          | 79.83     |
|                 | Memory block size | 4-dim. blocks | 72.07*         | 77.77*    |
|                 |                 | 16-dim. blocks  | 76.96*         | 80.89*    |
|                 |                 | 32-dim. blocks  | 75.39          | 79.6*     |
|                 |                 | 16-dim. blocks adj. | 79.27*         | 81.55*    |
|                 |                 | 32-dim. blocks adj. | 80.87*         | 81.64*    |
|                 | Memory block init. | Normal init. | 71.53*         | 76.92*    |
|                 |                 | Uniform init.   | 67.51*         | 69.23*    |
|                 |                 | Dense matched init. | 73.25*         | 78.41*    |
|                 | Replay format   | Image replay    | 67.02*         | 72.60*    |
| Toybox          | Unablated      | Ours            | 58.26          | 69.30     |
|                 | Replay format   | Image replay    | 52.33*         | 64.24*    |
| iLab            | Unablated      | Ours            | 63.41          | 71.56     |
|                 | Replay format   | Image replay    | 55.36*         | 58.14*    |
| CIFAR100        | Unablated      | Ours            | -              | 42.97     |
|                 | Replay format   | Image replay    | -              | 44.30*    |
| ImageNet        | Unablated      | Ours            | -              | 29.84     |
|                 | Replay format   | Image replay    | -              | 49.8*     |

Table S2. Top-1 accuracy on all tasks after completion of stream learning for supplementary model analysis experiments. Each value in the right-most two columns is the mean percentage top-1 accuracy across 5 independent runs, each with its own independent pretraining run on ImageNet. * denotes significant differences from Ours, with comparisons only done within each dataset (see Sec. S4). Results from Ours and Image replay on CORe50 are duplicated from Table 2 in the main paper. Online-CIFAR100 and Online-ImageNet are abbreviated as CIFAR100 and ImageNet respectively. The class-instance protocol is not applicable to either of these image datasets, because unlike CORe50, Toybox, and iLab, they do not consist of video clips. The highest accuracy within each dataset/training protocol combination is in bold.
size perturbations above. However, increasing the length of the memory blocks from 8 to 16 or 32 means that only half or one-quarter as many blocks respectively are needed to reconstruct each feature map, so only half/one-quarter as many indices need to be stored in the replay buffer per example. This allows double/quadruple the number of examples to be stored in the buffer within the same computer memory budget. When we allowed the maximum number of examples stored in the replay buffer to change accordingly (2n for 16-dim. blocks adj., 4n for 32-dim. blocks adj.), we observed accuracies that exceed those of Ours: 16-dim. blocks adj. achieves 2.5% and 1.8% higher accuracy than Ours on class-instance and class-iid respectively, and 32-dim. blocks adj. achieves 4.1% and 1.8% higher accuracy. During hyperparameter tuning for our main results, we observed that 16-dimensional memory blocks maximized testing accuracy.

S2.3. Initialization of the memory blocks matters

Our experiments suggest that our algorithm’s performance is somewhat sensitive to the initialization of the values in the memory blocks. CRUMB trains its memory blocks in tandem with network weights after initialization, and concatenates them in different combinations to reconstruct feature maps produced by an intermediate network layer. We compared stream learning performance of four codebook matrix initialization strategies, including initializing with values drawn from (1) a standard normal distribution (Normal init.), (2) a uniform distribution on the interval [0, 1] (Uniform init.), (3) a distribution designed to match that of the non-zero values in the feature maps to be reconstructed, with 64% of all values reset to zero to approximately match the sparsity of typical feature maps (Ours; see Sec. 4 in main paper), and (4) the same as (3), but with no values set to zero (Dense matched init.). Accuracy for Normal init. was 5.3% and 2.9% lower than Ours for class-instance and class-iid protocols respectively, accuracy for Uniform init. was 9.3% and 10.6% lower, and accuracy for Dense matched init. was 3.6% and 1.4% lower (see Table S2). It appears that drawing initial values for the memory blocks from a similar distribution to that of natural feature maps improves performance. When applying CRUMB to new network architectures, a simple alternative procedure to initialize the memory blocks would be to obtain feature maps from a batch of images, pool all values from all feature maps into one long vector, and initialize each memory block value by randomly drawing a value from this vector.

S2.4. Some memory blocks are coarsely interpretable

Visualizations of image locations where specific memory blocks are activated (Fig. S2) show that some memory blocks appear to be human-interpretable. Some blocks responded to features seen in images of one specific class or of a subset classes, and others responded to features
that are likely irrelevant to classification. In addition to the blocks visualized in Fig. S2, we found blocks that tend to respond to vertical lines, crosshatch patterns on balls and cups, pure white backgrounds, vegetation backgrounds, and wooden floor backgrounds, each of which can be interpreted as a semantic, compositional part of various test set images.

The procedure for generating the visualizations in Fig. S2 can be understood as follows. Test set images are first passed through the early layers of a convolutional neural network to produce a feature map, which CRUMB then reconstructs by concatenating memory block vectors to produce an approximated version of the original feature map (see Sec. 4.2 in main paper). In this study, each feature map is of size $13 \times 13 \times 512$, meaning spatial dimensions of $13 \times 13$ with 512 features at each spatial location. Each memory block is one of 256 row vectors in the $256 \times 8$ codebook matrix used for this analysis. The memory blocks are 8-dimensional vectors, so each spatial location in the feature map’s $13 \times 13$ grid is represented by a $512$-dimensional vector formed by concatenating $512/8 = 64$ memory blocks end-to-end. Fig. S2 shows at most one block per spatial location, the one activated by the first 8 features in the 512-dimensional feature vector, even though 64 memory blocks are activated at each location in total. We focus on the first 8 features for visualization purposes, because it is not necessarily the case that blocks activated by the first set of 8 features encode the same image features as they might when activated by the $k^{th}$ set of 8 features (where $2 \leq k \leq 64$). Finally, to produce the images in Fig. S2, each test set image is divided into a square $13 \times 13$ grid. Image grid locations are overlaid with colored squares, such that the color of each square depends on the memory block activated by the first 8 features at the corresponding spatial location in the feature map reconstructed by CRUMB. We only assigned colors to a handful of memory blocks with interesting properties, and sets of memory blocks that seemed to respond to very similar features were assigned the same color.

S3. Replay buffer size calculations

CRUMB, as well as other replay methods such as iCARL [41], REMIND [14] and Gradient Episodic Memory [31], store training examples in a replay buffer and play them back to the model during later tasks to prevent catastrophic forgetting. Models such as iCARL and GEM store pixel values of raw images, while REMIND and CRUMB store compressed representations of feature maps that take up less space per training example, allowing many more examples to be stored in a buffer within a fixed computer memory budget. Our main results compare the performance of CRUMB to that of various baseline algorithms under the assumption of a fixed computer memory budget for replay methods (we ignore this constraint for regularization methods, such as Elastic Weight Consolidation [20]). To calculate the maximum number of training examples we can store in the replay buffer for each experiment, we first set the number of examples $n_r$ that raw-image replay methods such as iCARL may store, then calculate how many examples ($n$) CRUMB can fit into the same amount of computer memory by the following formula:

$$n = \frac{n_r(3w_ih_i) - bd}{swh/d}$$  \hspace{1cm} (5)$$

Where $w_i$ and $h_i$ are raw image width and height respectively ($224 \times 224$ for our experiments), the codebook matrix has dimensions $b \times d$, and the feature map has dimensions $s \times w \times h$. The numerator corresponds to the number of 8-bit RGB values needed to store one image (minus a discounting factor for the number of values in the memory blocks themselves), and the denominator corresponds to the number of 8-bit integer indices required to encode one feature map.

For direct comparisons in our main results, we applied both CRUMB and REMIND to the SqueezeNet network architecture [17]. To calculate $n$ for REMIND, we multiplied the compression ratio provided by the REMIND paper ($959,665$ feature maps/10,000 raw images) by the ratio of values in one feature map from ResNet18 (used in the REMIND paper, $512 \times 7 \times 7$) to those in one feature map from SqueezeNet ($512 \times 13 \times 13$) [14]. We then multiplied the resulting ratio of $278,246$ feature maps/10,000 raw images by $n_r$ to obtain the corresponding $n$ for each dataset.

S4. Data analysis

S4.1. Data cleaning

For our main results on the video datasets CORe50, Toybox, and iLab, we noticed that a small subset of runs for some models had markedly reduced accuracy on the first task compared to other runs. To facilitate fair comparisons among models, we excluded all runs with an initial task accuracy less than 80% from all analysis and results. For the small number of algorithm/dataset/protocol combinations for which no runs exceeded 80% on the first task, we filtered at a 60% threshold, or a 40% threshold if no runs exceeded 60%. We did not encounter this issue for any runs of CRUMB on any dataset, or for any algorithm on the image datasets Online-CIFAR100 and Online-Imagenet.

S4.2. Statistics for model analysis experiments

Our model analysis experiments in Sec. 5.4 of the main paper and Sec. S2 in the supplementary material compared the performance of CRUMB with various ablated or otherwise perturbed versions of CRUMB. For each comparison with the original algorithm, we evaluated
statistical significance of pairwise differences using the following method:

1. Divide the test set from the dataset being used into batches of 100 images. The images should be randomly sampled without replacement, and the sampling should be done only once (or, using a fixed random seed) for all experiments such that each version of the algorithm is evaluated on the exact same batches of images.

2. Evaluate CRUMB and each experimentally perturbed version of CRUMB on the same set of image batches, recording mean top-1 accuracy on each batch. This is done for each of 5 independent training runs, and accuracies are pooled across runs. Therefore, for each training protocol (class-instance and class-iid, for which all analyses are kept separate), each version of the algorithm has $n_r \times n_b$ accuracy estimate values, where $n_r$ is the number of runs and $n_b$ is the number of 100-image batches in the test set. Conceptually, we treat the accuracy on each batch as an independent sample indicating the accuracy of the corresponding algorithm on a roughly continuous scale, with each run of each algorithm tested on the exact same batches of images.

3. Perform a paired-samples t-test for each comparison, using accuracy on each image batch of CRUMB and the perturbed version of CRUMB as a sample pair and pooling sample pairs across runs. We used a global p-value cutoff of $p < 0.01$ to report the statistical significance of t-test results for each comparison between CRUMB and a perturbed version of CRUMB.