Tuned Compositional Feature Replays for Efficient Stream Learning

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

***Abstract*— Our brains extract durable, generalizable knowledge from transient experiences of the world. Artificial neural networks come nowhere close to this ability. When tasked with learning to classify objects by training on nonrepeating 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 combining generic parts. CRUMB concatenates trainable and reusable "memory block" vectors to compositionally reconstruct feature map tensors in convolutional neural networks (CNNs). Storing the indices of memory blocks used to reconstruct new stimuli enables memories of the stimuli to be replayed during later tasks. This reconstruction mechanism also primes the neural network to minimize catastrophic forgetting by biasing it toward attending to information about object shapes more than information about image textures and stabilizes the network during stream learning by providing a shared feature-level basis for all training examples. These properties allow CRUMB to outperform an otherwise identical algorithm that stores and replays raw images while occupying only 3.6% as much memory. We stress-tested CRUMB alongside 13 competing methods on seven challenging datasets. To address the limited number of existing online stream learning datasets, we introduce two new benchmarks by adapting existing datasets for stream learning. With only 3.7%–4.1% as much memory and 15%–43% as much runtime, CRUMB mitigates catastrophic forgetting more effectively than the state-of-the-art. Our code is available at https://github.com/MorganBDT/crumb.git**

***Index Terms*— Brain-inspired replay, catastrophic forgetting, deep learning, stream learning.**

Manuscript received 6 March 2023; revised 25 October 2023 and 24 November 2023; accepted 3 December 2023. Date of publication 25 December 2023; date of current version 6 February 2025. This work was supported in part by NIH under Grant R01EY026025; in part by the National Research Foundation, Singapore, through its AI Singapore Programme (AISG) under Award AISG2-RP-2021-025 and its Singapore National Research Foundation Fellowship (NRFF) under Award NRF-NRFF15-2023-0001; in part by the Center for Brains, Minds and Machines through the NSF Science and Technology Center Award under Grant CCF-1231216; and in part by the National Institute of General Medical Sciences under Award T32GM007753 and Award T32GM144273. *(Morgan B. Talbot and Rushikesh Zawar contributed equally to this work.) (Corresponding authors: Mengmi Zhang; Gabriel Kreiman.)*

Morgan B. Talbot is with the Harvard-MIT Program in Health Sciences and Technology, Cambridge, MA 02139 USA, also with Boston Children's Hospital (BCH), Boston, MA 02115, USA, and also with the Center for Brains, Minds, and Machines (CBMM), Cambridge, MA 02142 USA.

Rushikesh Zawar is with Harvard University, Cambridge, MA 02138 USA, also with Boston Children's Hospital (BCH), Boston, MA 02115 USA, and also with the Birla Institute of Technology and Science at Pilani (BITS Pilani), Pilani 333031, India.

Rohil Badkundri is with Harvard University, Cambridge, MA 02138 USA, also with Boston Children's Hospital (BCH), Boston, MA 02115 USA, and also with the Center for Brains, Minds, and Machines (CBMM), Cambridge, MA 02142 USA.

Mengmi Zhang is with the Center for Brains, Minds, and Machines (CBMM), Cambridge, MA 02142 USA, also with Boston Children's Hospital (BCH), Boston, MA 02115 USA, also with the Agency for Science, Technology and Research (A*STAR) Center for Frontier AI Research (CFAR), Singapore 138632, also with the A*STAR Institute for Infocomm Research (I2R), Singapore 138632, and also with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: mengmi.zhang@ntu.edu.sg).

Gabriel Kreiman is with Boston Children's Hospital (BCH), Boston, MA 02115, USA, also with the Harvard Medical School, Boston, MA 02115 USA, and also with the Center for Brains, Minds, and Machines (CBMM), Cambridge, MA 02142 USA (e-mail: gabriel.kreiman@tch.harvard.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNNLS.2023.3344085, provided by the authors.

Digital Object Identifier 10.1109/TNNLS.2023.3344085

I. Introduction

HUMANS adapt to new and changing environments by learning rapidly and continuously. Previously learned skills and experiences are retained even as they are transferred and applied to new tasks, which are learned from a stream of highly temporally correlated stimuli and without direct access to past experiences. In contrast, in standard class-incremental image classification settings in continual learning, neural networks are presented with images that are independently and identically distributed (i.i.d.), with multiple presentations of each image [1], [2], [3]. To better emulate a human learning environment or that of an autonomous robot that must learn in real time, we focus on a challenging and realistic variant of class-incremental learning—*online stream learning*. Online stream learning has two key characteristics (Fig. 1): 1) the input is in the form of video streams with highly temporally correlated frames and 2) each training example is presented only once: no repeated presentations of old data are allowed.

In online stream learning settings, current machine learning systems tend to fail to retain good performance on previously learned tasks, exhibiting catastrophic forgetting [4], [5], [6]. Catastrophic forgetting is a pervasive problem in continual learning settings for both deep neural networks [7] and other models such as linear regression [8] and self-organizing maps [9], and can also cause neural models to be biased toward more recently encountered training data [10]. One strategy for overcoming catastrophic forgetting is to store a copy of all or most encountered training examples for later replay, effectively converting to an offline learning paradigm [11]. This approach, however, often requires an impractically large amount of memory [12]. Moreover, much of the information in raw images is redundant, with many pixel values needed to represent each feature-level concept relevant to classification. Finally, storing old training data might also be undesirable from a data security or privacy standpoint, such as in hospitals and other healthcare settings [13].

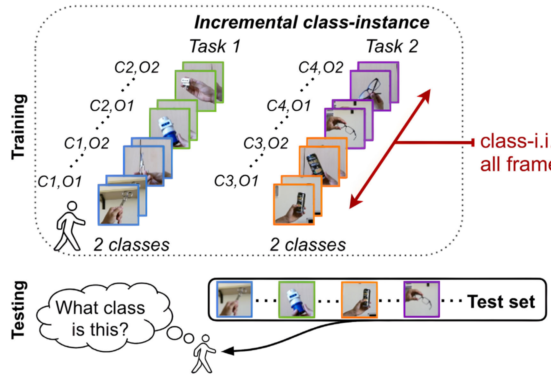


Fig. 1. Schematic of online stream learning protocols. For each task, the model learns to classify a set of new classes (C1, C2, and so on in the figure) while training on video clips of several objects from each class (O1 and O2) for only one epoch. During testing, the model has to classify images from all seen classes without knowing task identity. In the class-instance training protocol, the order of video clips is shuffled, but the order of frame images is preserved within each clip. In the class-i.i.d. training protocol, all images within each task are randomly shuffled. Class-i.i.d. is the only option for datasets such as ImageNet that consist of standalone images and not video clips.

To address both memory inefficiency and data privacy concerns while achieving state-of-the-art online stream learning performance, we propose a new continual learning approach, compositional replay using memory blocks (CRUMB) (Fig. 2). In our method, each new image is processed by the early layers of a convolutional neural network (CNN) to produce a feature map tensor. The feature map is decomposed by slicing it into chunks, each of which is a vector of feature activations at a specific spatial location. Each chunk is then replaced by the most cosine-similar row ("memory block") of a trainable "codebook matrix." This mechanism encodes images as a composition of discrete feature-level concepts, some of which appear to have semantic interpretations. Storage of a complete training example for replay requires keeping only the indices of the memory blocks needed to reconstruct the original feature map, along with the class label, occupying only 3.6% of the memory footprint of a raw image. During replay, feature maps reconstructed via stored indices are fed to the later layers of the CNN such that these layers are trained on both stored and newly encountered images to learn new tasks while retaining previous knowledge.

Our key contributions are given as follows.

1) *Trainable Compositional Replay:* We propose a new compositional feature-level replay algorithm, CRUMB, for online stream learning. The composition mechanism is end-to-end trainable and reusable. CRUMB's codebook of memory blocks captures the essential components needed for reconstructing feature maps. During the pretraining phase, the memory block mechanism primes the CNN for stream learning with high accuracy and induces a beneficial bias toward object shapes. Using memory blocks as a shared basis for new and recalled examples helps stabilize the network during stream learning.
2) *Reduced Forgetting:* We tested CRUMB on seven continual learning datasets alongside 13 competing methods, showing that CRUMB typically outperforms state-of-the-art approaches by large margins.
3) *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% as much memory. In addition, compared with the next most accurate method (REMIND [4]), CRUMB requires only about 15%–43% as much training runtime and occupies only 3.7%–4.1% of REMIND's peak memory footprint.
4) *New Benchmarks:* We adapted two datasets, Toybox [14] and iLab [15], to introduce new online stream learning benchmarks. All benchmark details along with source code, results, and data are available at https://github.com/MorganBDT/crumb.git.

II. Related Work

A. Weight Regularization

Weight regularization methods typically store weights trained on previous tasks and impose constraints on subsequent weight updates to minimize catastrophic forgetting [12], [16], [17], [18], [19], [20], [21]. However, storing the importance of the millions of parameters required by state-of-the-art recognition models across all previous tasks is costly [12], [22]. Moreover, empirical comparisons suggest that weight regularization methods typically do not mitigate catastrophic forgetting as effectively as architecture adaptation and replay methods [23].

B. Architecture Adaptation

Architecture adaptation methods expand or reorganize the structure of their neural networks to accommodate new tasks to be learned. Approaches include adding groups of new neurons (which does not always scale well) [12], [16], [18], [19], [20], [24], isolating parts of a larger neural network for each task [25], [26], [27], [28], [29], compressing parameters in a consolidation phase [30], and pruning neurons or weights for later reuse. Pruning approaches include $L1$ regularization and activity threshold-based sparsification of neurons [31], and combining pruning of weights with parameter importance-based regularization [32]. Neuron pruning/reuse can also be combined with the addition of new neurons to improve performance and enable increased flexibility [33]. All of these approaches add significant complexity, and some require explicit labeling of task identities, which is not feasible in many online learning applications.

C. Image and Feature Replay

In replay methods, images or features from previous tasks are stored and later retrieved or regenerated to be shown to the model to prevent forgetting [17], [34], [35], [36], [37], [38], [39]. Replay can be highly effective but comes with some caveats. Relying on replaying limited sets of stored examples can lead to overfitting. Storing a large number of raw images for replay is memory-intensive. To limit memory requirements, generative replay systems combine data from new tasks with

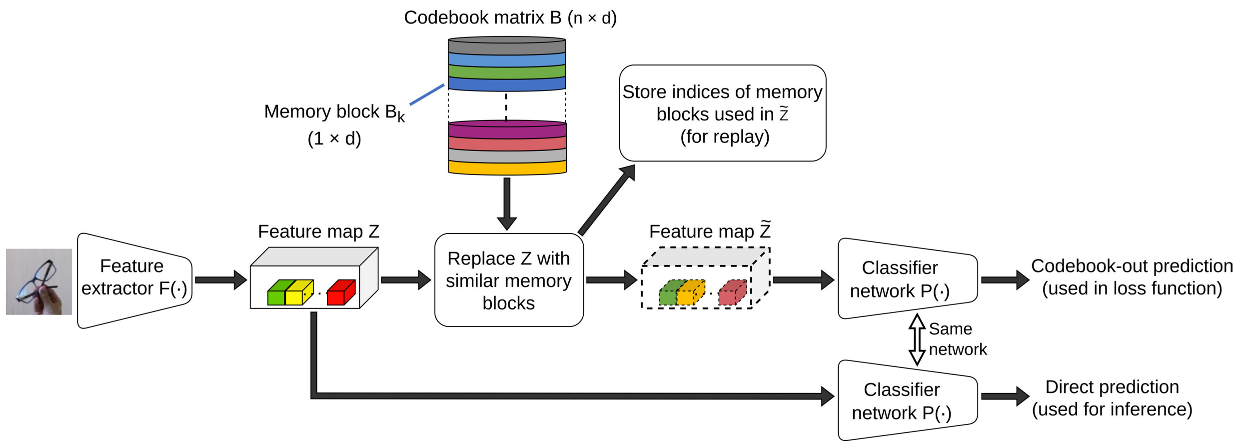


Fig. 2. Schematic of CRUMB, the proposed algorithm for online stream learning. The model consists of a CNN ($\mathbf{F}(\cdot)$ for early layers and $\mathbf{P}(\cdot)$ for later layers) and a codebook matrix $\mathbf{B}$ used for compositional reconstruction of feature-level activation tensors (feature maps $\mathbf{Z}$). Each row $\mathbf{B_k}$ of $\mathbf{B}$ is a "memory block" vector. CRUMB uses the feature extractor $\mathbf{F}(\cdot)$ to produce an initial feature map and then determines which memory blocks to retrieve from $\mathbf{B}$ based on a cosine-similarity addressing mechanism. The feature maps reconstructed from the memory blocks ($\tilde{\mathbf{Z}}$), and the original feature maps ($\mathbf{Z}$) are used to obtain separate classification losses from the same classifier network $\mathbf{P}(\cdot)$ ("**codebook-out loss**" and "**direct loss**"). Only codebook-out loss is used for weight updates during stream learning, although the two losses are added in a weighted sum to calculate the total loss during pretraining. To avoid catastrophic forgetting, we store the row indices of retrieved memory blocks along with class labels, for example, images from each task. In later tasks, following each batch of new images, we "replay" a batch of old feature maps to $\mathbf{P}(\cdot)$ after reconstructing them using stored memory block indices.

synthetic data produced by generative models to resemble previously encountered stimuli [40], [41], [42], [43], [44], [45], [46]. However, the generative models needed to create adequate synthetic data remain large, memory-intensive, and difficult to train [22].

Other replay methods save memory by storing raw or compressed feature maps from intermediate CNN layers [4], [47] or generate synthetic examples by sampling from simple feature-level probability distributions for each class [48]. REMIND [4] achieves high performance in online stream learning by compressing feature maps using a product quantizer [49]. However, the product quantizer is trained by performing k-means clustering on a large subset of training data stored in memory, a process that scales poorly for increasingly large datasets. In contrast, CRUMB's differentiable codebook is trained by backpropagation alongside other network parameters, dramatically reducing memory requirements for codebook initialization.

CRUMB's feature-based replay mechanism is inspired by biological replay observed in the hippocampus and other brain areas [50], [51], [52] and by complementary learning systems theory [53]. Recent work has explored modeling the hippocampus and neocortex as separate neural networks that interact via distillation losses and other mechanisms [54]. In contrast to storing knowledge implicitly in a short-term memory network, CRUMB's memory blocks and replay buffer store short-term memories that represent individual training examples and interact with the CNN via replay to facilitate long-term memory storage and consolidation.

III. Methods

A. Online Stream Learning Benchmarks

1) Training Protocols: We consider two online class-incremental settings: class instance and class-i.i.d. [4] (Fig. 1).

a) Class instance: Each task contains short video clips of different objects from two or more classes, and the video clips are presented one after another in random order within each task without repetition. 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 highly correlated images of each object before moving on to the next.

b) Class-i.i.d.: Images/video frames are randomly shuffled within each task but not interspersed among tasks and are shown only once like in class instance. This is a less challenging protocol and should not be considered stream learning in the strictest sense because the shuffling of images in each task destroys any temporal structure among images.

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 subsequent tasks only once. This emulates real-time acquisition of training data that cannot be stored except in a limited-capacity replay buffer.

2) Stream Learning Benchmark Datasets: We evaluated our model on five video datasets (class-instance and class-i.i.d. protocols) and two image datasets (class-i.i.d. only). For all datasets, we used different task and example orderings across training runs. A global holdout test set of images/frame sequences was used for all runs. To help address the limited number of online stream learning benchmarks, we adapted two datasets designed for studying object transformations, Toybox [14] and iLab [15], for online stream learning.

The **CORe50 video dataset [55]** contains images of 50 objects in ten classes. Each object has 11 instances, which are 15-s video clips of the object under particular conditions and poses. We followed [4] for the training and testing data split and sampled each video at 1 frame per second (FPS).

The **Toybox video dataset [14]** contains videos of toy objects from 12 classes. We used a subset of the dataset containing 348 toy objects, each of which has ten instances containing different patterns of object motion. We sampled each instance at 1 FPS, resulting in 15 images per instance per object. We chose three of the ten instances for our test set, leaving seven instances for training.

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

The **iCub (iCubWorld Transformations) video dataset [56]** contains videos taken by the iCub robot of 20 classes of household objects undergoing viewpoint transformations. We used isolated rotation, scaling, and background transformations as our training set and the provided "MIX" sequence (which combines all transformations) as our test set.

The **iLab + CORe50 video dataset** combines iLab and CORe50 to create a stream learning benchmark with 24 distinct classes. All iLab classes are learned before CORe50, introducing a mild domain shift. We uniformly subsample iLab to balance the number of images per class with CORe50.

To evaluate our model in long-range online class-incremental learning with many more classes than the video datasets described above, we also include results on two image datasets. The standard **Online-CIFAR100 image dataset [57]** is split into 20 tasks with five classes each, while the standard **Online-Imagenet image dataset [58]** is split into ten tasks with 100 classes each. Class-instance training is not applicable to image datasets because they do not consist of videos.

3) Baseline Algorithms for Comparison: All baseline algorithms use the same training protocols as CRUMB. CRUMB and most baselines use a SqueezeNet CNN pretrained on ImageNet [59], but due to implementation constraints, adaptive aggregation network (AAN) [60], CoPE [61], GSS [36], LwF [16], RM [39], and Stable SGD [62] use non-pretrained ResNet models [63]. We reimplemented some methods due to varying code availability. CRUMB and all baselines are implemented using the PyTorch library [64].

a) Weight regularization: We compared against elastic weight consolidation (EWC) [12], synaptic intelligence (SI) [19], memory aware synapses (MAS) [65], learning without forgetting (LwF) [16], and Stable SGD [62].

b) Memory distillation and replay: We compared against gradient episodic memory (GEM) [38], incremental classifier and representation learner (iCARL) [35], bias correction (BiC) [34], gradient sample selection (GSS) [36], continual prototype evolution (CoPE) [61], AAN [60], REMIND [4], and rainbow memory (RM) [39].

The **lower bound** is trained sequentially over all tasks without any measures to avoid catastrophic forgetting.

The **upper bound** is trained offline on shuffled images from both the current and all previous tasks over multiple epochs.

Chance predicts class labels by randomly choosing one out of the total of C_t classes seen in or before current task t.

B. Proposed Algorithm: CRUMB

We propose a new continual learning algorithm, CRUMBs. CRUMB consists of a 2-D CNN augmented by an $n \times d$ codebook matrix B. A schematic of CRUMB is shown in Fig. 2, with further details described in algorithm 1. CRUMB extracts a feature map from each given image using the early layers of a pretrained CNN and stores a subset of the feature maps in a buffer. When CRUMB later encounters a new task, it avoids catastrophic forgetting of previous tasks by replaying stored feature maps of images from those tasks to the later layers of the network. To reduce memory requirements, CRUMB uses its codebook matrix B to reconstruct each feature map. Rows of B ("memory blocks") are concatenated to form tensors that approximate the original feature maps, and only the indices of activated memory blocks need to be stored to enable later reconstruction. All computations from memory block reconstruction forward are differentiable, allowing B to be learned alongside the CNN weights.

Algorithm 1 CRUMB at Task t

Input: training images I_t from new classes, stored codebook matrix B, replay buffer X of stored memory block indices and their class labels (maximum number n_X of total stored examples in X varies by dataset).
Training:
for batch **in** I_t **do**
  Reconstruct feature map Z as $\tilde{Z}$ for each image in batch by concatenating memory blocks (rows of B)
  Train $P(\cdot)$ using loss $L(P(Z), P(\tilde{Z}), y_c)$, with $\alpha = 0$ in streaming
  Train memory blocks in B that form part of $\tilde{Z}$, using $L_{CE}(P(\tilde{Z}), y_c)$
  if $t > 1$ **then**
    Randomly sample images x out of X to form a replay batch
    Reconstruct $\tilde{Z}$ for each x by concatenating memory blocks
    Train $P(\cdot)$ using loss $L_{CE}(P(\tilde{Z}), y_c)$
    Train memory blocks in B that are part of $\tilde{Z}$ via backpropagation
  end if
end for
Store in X: memory block indices for reconstruction of every j^{th} image
Testing:
for batch **in** testing images **do**
  Compute predictions $p = P(F(\cdot))$ on test images using Z only
end for

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 [66], the parameters of $F(\cdot)$ are pretrained for image classification using ImageNet [58] 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, and height). Z is reconstructed using B to form $\tilde{Z}$, and a class prediction output can then obtained as $P(\tilde{Z})$. The parameters of $P(\cdot)$ are initially pretrained alongside $F(\cdot)$ on ImageNet using standard methods but also undergo additional ImageNet pretraining with an objective incorporating predictions on both Z and $\widetilde{Z}$. Prior to stream learning, only the final layer of $P(\cdot)$ is randomly reinitialized to reflect the number of classes to be learned during streaming.

2) Reconstructing Feature Maps From Memory: CRUMB produces reconstructed feature map $\widetilde{Z}$ 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. Hyperparameters n and d are determined empirically (see Sections IV-D4 and IV-D5). Z is first partitioned evenly along its feature dimension into s/d tensors, with each tensor Z_f of size $d \times w \times h$. Each tensor Z_f is further partitioned by spatial location into $w \cdot h$ vectors, denoted $Z_{f,x,y} \in \mathbb{R}^{\mathrm{d}}$, where d is also the dimension of each row B_k in the matrix B. For each vector $Z_{f,x,y}$ in Z, a similarity score γ_k is calculated between it and each memory block B_k as follows:

$$\gamma_{f,x,y,k} = \left\langle Z_{f,x,y}, \frac{B_k}{\|B_k\|_2} \right\rangle \tag{1}$$

where $\langle\cdot,\cdot\rangle$ is the dot product and $\|\cdot\|_2$ is the $L2$-norm. 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 γ similarity value replaces $Z_{f,x,y}$ at its corresponding location in $\widetilde{Z}$ as follows:

$$\widetilde{Z}_{f,x,y} \leftarrow B_{k_{f,x,y}}, \quad \text{where } k_{f,x,y} = \underset{k}{\operatorname{argmax}}(\gamma_{f,x,y,k}). \tag{2}$$

Because $\widetilde{Z}$ is reconstructed entirely from memory blocks B_k, we can save all information needed to reconstruct $\widetilde{Z}$ again later by storing both B and the values of k at each f, x, and y location in $\widetilde{Z}$. Thus, the feature map for the ith training image can be stored as $m_i = (k_{1,1,1}, \ldots, k_{f,x,y}, \ldots, k_{s/d,w,h})$.

For example, in our main implementation, Z is a $512 \times 13 \times 13$ tensor. $d = 16$ so that Z is split into $32 \times 13 \times 13 = 5408$ vectors of length 16, which are each replaced in $\widetilde{Z}$ by a 16-D memory block from a 256×16 matrix B. The memory blocks themselves occupy a near-negligible amount of memory: $256 \times 16 = 4096$ floating-point values, compared to the 5408 integers required to store a single compressed training example in this implementation.

3) Training: During training, both Z and $\widetilde{Z}$ are passed separately through the classifier $P(\cdot)$ to obtain two classification probability vectors $p = P(Z)$ and $\widetilde{p} = P(\widetilde{Z})$, where the dimension of p_t and $\widetilde{p}_t$ is equal to the total number of classes C_t that have been seen in or before 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 $\widetilde{p}$. With y_c defined as the ground-truth class label of a given image

$$L(p, \widetilde{p}, y_c) = \alpha L_{\mathrm{CE}}(p, y_c) + \beta L_{\mathrm{CE}}(\widetilde{p}, y_c). \tag{3}$$

Larger values of α penalize "direct" prediction errors from $P(Z)$, while larger values of β penalize "codebook-out" prediction errors from $P(\widetilde{Z})$. Although our model generates class predictions based on both Z and $\widetilde{Z}$, 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 ($\alpha = 1$ and $\beta = 1$) and then removing the direct loss for stream learning ($\alpha = 0$ and $\beta = 1$) (see Section IV-D6) for analysis). Setting α to 0 makes the loss function for new batches of images more similar to that used for replay, where only $\widetilde{Z}$ is available. Replacement of Z by the reconstructed version $\widetilde{Z}$ can be viewed as both a means to efficiently mitigate catastrophic forgetting and a regularization technique to prevent overfitting and stabilize the CNN.

Although values in CRUMB's memory blocks play the role of activation values in their reconstruction of $\widetilde{Z}$, they are trainable parameters of the model. Backpropagation from $\widetilde{Z}$-based predictions generates gradients for the values in each memory block used for reconstruction, and stochastic gradient descent modifies the memory blocks toward minimizing the same training objective used for the network weights (cross-entropy loss).

4) Initializing the Codebook Matrix and CNN: CRUMB's performance benefits from targeted initialization and pretraining of its CNN and the memory blocks in its codebook matrix, especially in the class-instance setting. The values in the codebook matrix B directly replace those in "natural" feature maps derived from images during training—accordingly, B is initialized using a simple univariate distribution designed to match that of natural feature maps from a pretrained network. In early experiments, we tried initializing the codebook using k-means cluster centers from feature vectors of CIFAR100 images [57], but this did not improve the performance.

Stream learning performance was substantially improved by pretraining CRUMB on ImageNet [58] classification with 1000 classes, compared to applying CRUMB to stream learning with a CNN pretrained by standard methods. Pretraining tunes the values in the memory blocks and also regularizes the CNN in preparation for stream learning by training it to make predictions using lossy reconstructions of feature maps.

5) Replay to Mitigate Catastrophic Forgetting: In online stream learning (see Section III-A1), the model is presented with images I_t from new classes c^{new} in task t, where c^{new} are drawn from the subset of classes in the dataset that the model has not seen in previous tasks.

Replay of examples from previous tasks is a proven strategy to mitigate catastrophic forgetting in class-incremental settings [17], [34], [35], [36], and feature-level replay can be considerably more memory-efficient than storing raw images [4]. 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.

CRUMB stores up to n_X pairs of labels and tensors (y_i, m_i), corresponding to images from old classes c^{old} of previous tasks. Depending on the number of seen classes C_{t-1}, the storage for each old class contains n_X/C_{t-1} pairs. n_X is chosen for each dataset depending roughly on the total number of classes. Some algorithms select representative image examples to store and replay based on different scoring functions [67], [68], [69], [70]. However, random sampling uniformly across classes yields outstanding performance in continual learning tasks [22], and we adopt this strategy to select examples from the buffer for replay. To choose training examples for storage

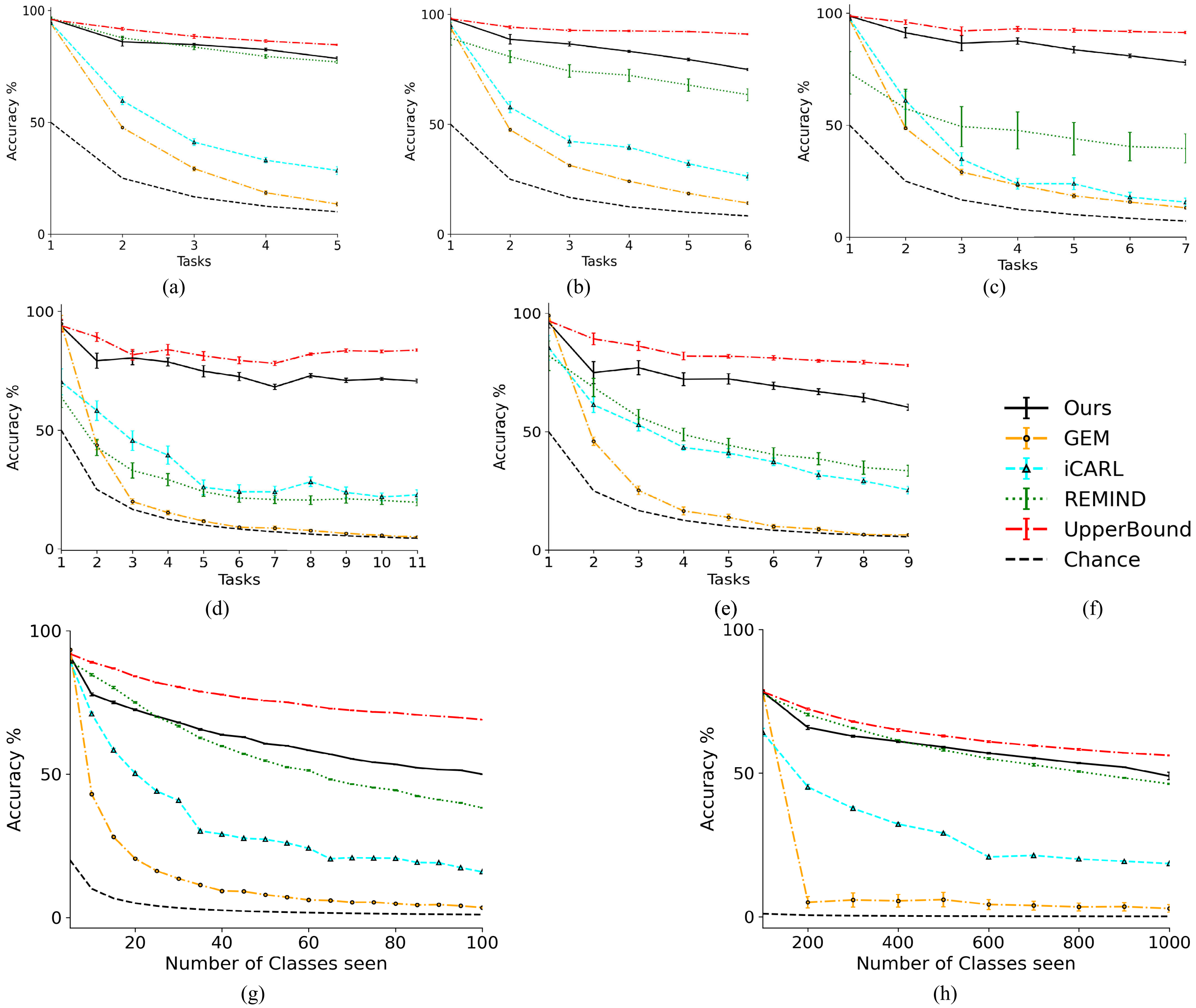


Fig. 3. CRUMB outperforms most baseline algorithms and approaches the upper bound on some datasets. Line plots show top-1 accuracy in online stream learning on video datasets: (a) CORe50, (b) Toybox, (c) iLab, (d) iLab + CORe50, and (e) iCub in the class-instance training protocol (class-i.i.d. plots are in Supplementary Fig. S1), and image datasets: (g) Online-CIFAR100 and (h) Online-ImageNet (class-i.i.d.). (f) Legend. 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 from ten runs (five runs for ImageNet), where each run has different class and image/video clip orderings. Error bars show the root-mean-square error (RMSE) among runs. The results for all baselines are in Table I.

in the replay buffer, CRUMB keeps every jth image in each batch, where j is calculated by dividing the number of training images in each task by the replay buffer capacity n_x such that the buffer is filled near the end of the current task's training epoch. In the class-instance setting, this maximizes sample diversity by minimizing the number of frames sampled from within the same video clip and further avoiding sampling frames from the same clip that are in close temporal proximity.

For replay-based baseline methods (iCARL, REMIND, and so on), 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 (see details in Supplementary Section S5). Aside from n_x and the training batch size (which is smaller for video datasets), CRUMB uses the same hyperparameters for all datasets, including n, d, α, β, learning rate, and initialization and pretraining protocols.

IV. Results

A. 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, i.i.d. fashion. This renders the benchmark equivalent to offline class-incremental learning ("upper bound" in Fig. 3) [11]. By storing a subset of old examples and using a compositional strategy to 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

TABLE I

CRUMB Outperforms State-of-the-Art Algorithms on Most Benchmarks. Each Number Is the Mean Top-1 Accuracy on All Tasks/Classes After the Completion of Stream Learning. Values Are Averaged From Ten (CORe50, Toybox, iLab, iCub, iLab + CORe50, and Online-CIFAR100) or Five (Online-ImageNet) Independent Runs. The Highest Accuracy in Each Column (Excluding the Offline Upper Bound) Is in Bold, While the Second-Highest Is Italicized. Algorithm Name Abbreviations Can Be Found in Section III-A3. Class Instance, in Which Video Frames Are Presented in Temporal Order, Is Only Applicable to Video Datasets CORe50, Toybox, iLab, iCub, and iLab + CORe50. Due to Resource Constraints, for Online-ImageNet, iCub, and CORe50 + iLab, We Tested a Subset of Baseline Algorithms. Class-Instance and Class-iid Settings Are Abbreviated as "inst" and "iid," Respectively. Results Are Grouped Vertically With Memory Distillation and Replay Methods at the Top (Ours—CoPE), Weight Regularization Methods in the Middle (EWC—LwF), and Lower/Upper Bounds at the Bottom. Data Preparation Methods Are Detailed in Supplementary Section S4.A

	CORe50 [55]		Toybox [14]		iLab [15]		iCub [56]		iLab+CORe50		CIFAR100 [57]	ImageNet [58]
	inst	iid	inst	iid	inst	iid	inst	iid	inst	iid	iid	iid
Ours	**78.5**	**81.2**	**74.9**	*75.7*	**77.9**	*79.5*	**60.0**	**65.8**	**66.0**	**74.4**	**49.9**	**48.9**
REMIND [4]	*77.0*	*76.0*	*66.2*	**84.1**	*48.1*	**81.0**	*33.2*	*58.5*	*22.9*	*67.2*	*38.2*	*46.2*
iCARL [35]	27.0	28.5	27.3	26.5	15.6	23.6	23.4	20.9	17.8	23.2	15.9	18.5
GEM [38]	11.9	13.5	14.3	15.7	13.0	12.8	5.2	6.1	4.6	4.5	3.5	2.9
RM [39]	12.0	12.4	9.8	20.8	18.2	9.3	-	-	-	-	4.2	-
AAN [60]	14.0	15.6	13.2	17.6	10.6	15.0	-	-	-	-	6.6	-
GSS [36]	15.0	15.6	14.7	15.0	13.0	12.8	-	-	-	-	3.2	-
BiC [34]	10.2	11.8	11.0	10.2	11.2	10.9	-	-	-	-	4.0	-
CoPE [61]	16.6	16.3	21.7	22.4	17.6	18.6	-	-	-	-	8.8	-
EWC [12]	12.2	12.4	14.3	15.7	13.5	13.0	7.3	6.4	11.7	12.0	3.9	0.1
MAS [65]	14.4	17.4	18.9	19.2	20.5	22.1	4.8	4.8	4.4	4.3	5.5	0.1
SI [19]	12.0	12.9	14.3	15.5	12.8	13.0	5.3	5.7	4.4	5.0	3.6	8.8
Stable SGD [62]	13.7	13.2	13.5	13.8	9.8	6.9	-	-	-	-	7.3	-
LwF [16]	12.5	12.4	21.9	20.9	10.5	11.9	-	-	-	-	4.2	-
Lower bound	12.1	12.8	15.5	16.9	12.8	16.4	5.8	6.0	4.5	4.6	3.5	3.0
Upper bound	85.3	84.6	91.0	92.0	91.3	91.4	76.9	78.0	83.1	81.7	69.0	56.1

a tiny 0.013% fraction (on CORe50) of the memory footprint. Accordingly, given a fixed memory budget, CRUMB outperforms all competing models in all five tested video stream learning datasets in the class-instance setting, often by large margins. For example, as shown in Fig. 3(a)–(e), CRUMB's top-1 accuracy on all tasks after class-instance stream learning exceeds that of REMIND by 1.5%, 8.7%, 29.8%, 26.8%, and 43.1%, iCARL by 51.5%, 47.6%, 62.3%, 36.6%, and 48.2%, and GEM by 66.6%, 60.6%, 64.9%, 54.8%, and 61.4% on CORe50, Toybox, iLab, iCub, and iLab + CORe50, respectively. The class-instance performance of all models is shown in Table I. CRUMB approaches the offline upper bound to within 6.8%, 16.1%, 13.4%, 16.9%, and 17.1% on the same datasets, demonstrating strong mitigation of catastrophic forgetting.

The less challenging class-i.i.d. 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 class-i.i.d. task are shuffled in an i.i.d. manner, removing the local temporal correlations introduced by sequential frames in video clips. As with class instance, CRUMB achieves excellent class-i.i.d. performance: as shown in Supplementary Fig. S1a–e, CRUMB's top-1 accuracy on all tasks after class-i.i.d. learning exceeds that of iCARL by 52.7%, 49.2%, 55.9%, 44.9%, and 51.2% and of GEM by 67.7%, 60%, 66.7%, 59.7%, and 69.9% on CORe50, Toybox, iLab, iCub, and iLab + CORe50, respectively. The class-i.i.d. performance of all models is shown in Table I. CRUMB approaches the offline upper bound to within 3.4%, 16.3%, 11.9%, 12.2%, and 7.3% on the same datasets. The performance of REMIND and CRUMB was comparable on class-i.i.d., with CRUMB's accuracy higher than REMIND's by 5.2%, 7.3%, and 7.2% on CORe50, iCub, and iLab + CORe50, respectively. However, REMIND's accuracy was higher than CRUMB's by 8.4% and 1.5% on Toybox and iLab, respectively.

On all benchmarks, CRUMB's closest competitor by far was REMIND, with all other methods exhibiting much lower accuracy. In general, the regularization baselines greatly underperformed the replay-based methods. This is perhaps due to limited exposure to each task given that each image may be visited only once and/or because of overfitting to temporally correlated data, especially in the class-instance setting: replay addresses both of these issues, while regularization methods generally do not. 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 and GEM. This increases the diversity of the replayed stimuli.

B. Stream Learning on Natural Image Datasets

Although stream learning of CORe50, Toybox, iLab, iCub, and iLab + CORe50 is highly challenging, these datasets have only 10–24 classes each. To demonstrate CRUMB's capacity for long-range stream learning of many classes, we employed standard image datasets Online-CIFAR100 and Online-ImageNet. CRUMB outperformed all baselines on both of these datasets (see Table I). On Online-CIFAR100, CRUMB's mean top-1 accuracy after class-i.i.d. stream learning exceeds that of REMIND by 11.7%, iCARL by 34%, and GEM by 46.4%, performing within 19.1% of the offline upper bound. On Online-Imagenet, CRUMB outperforms REMIND by 2.7%, iCARL by 30.4%, and GEM by 46%, performing within 7.2% of the offline upper bound [see also Fig. 3(g)–(h)].

TABLE II

CRUMB Uses Only 3.7%–4.1% of REMIND's Peak RAM Usage, and Its Runtime Is Approximately 15%–43% of REMIND's

Dataset	Peak RAM (GB)		Runtime (hours)	
	Ours	REMIND	Ours	REMIND
CIFAR100	**0.036**	0.87	**0.29**	1.91
Imagenet	**1.66**	44.34	**15.50**	35.64

C. Memory and Runtime Efficiency

CRUMB's closest competitor in top-1 accuracy is REMIND [4]. Both models require specific pretraining procedures. REMIND trains a product quantizer using k-means clustering of feature vectors, which requires a large portion of training data to be held in memory simultaneously. In contrast, CRUMB's codebook matrix is trained by backpropagation in tandem with CNN parameter updates. This approach requires only 3.7%–4.1% of the peak RAM usage of REMIND for large datasets such as Online-CIFAR100 and Online-Imagenet. CRUMB also has a runtime only about 15%–43% as long as REMIND's (Table II).

D. Model Analysis

To elucidate the importance of CRUMB's various components, we performed a series of ablation studies and experiments with altered training procedures. Accuracy results on CORe50 in both class-instance and class-i.i.d. settings are included for each experiment in Tables III and IV, but throughout the text in this section, we discuss class-instance results except where otherwise stated. Experiment names are in **bold** throughout this section. We conducted statistical significance tests for each experiment (see Supplementary Section S4.B).

1) Replay: n CRUMB Feature Maps Beat n Images: Feature-level replay is the main mechanism by which CRUMB prevents catastrophic forgetting. Removing replay dramatically reduces accuracy by 64.9%. However, CRUMB does not require storing a large number of feature maps to mitigate forgetting: reducing buffer size n_X from 200 (**Ours**) to 100 (**Half capacity**) and to 50 (**Quarter capacity**) had a relatively small impact, with 4.7% and 13.3% 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 dramatically higher accuracy than storing *the same number* n_X *of entire raw images* and training the network without any feature map reconstruction (**Image replay**), even though CRUMB's reconstruction of feature maps inevitably discards information and uses only 3.6% as much memory. As shown in Table III, **Ours** attains 10.4% higher accuracy than **Image replay** on CORe50 (buffer size $n_x = 200$), 4.2% higher on Toybox, 15.4% higher on iLab, 7.8% higher on iCub, and 2.8% higher on iLab + CORe50. This result appears to hold only for the five video streaming datasets, however: **Ours** attained accuracy 0.16% higher than **Image replay** on Online-CIFAR100 (not statistically significant) and 18.3% lower than **Image replay** on Online-ImageNet, in the class-i.i.d. setting. **Ours** uses only 3.6% as much memory as **Image replay**: when the amount of memory usage is held constant (e.g., in gigabytes) instead of the maximum number of replay buffer items n_x, CRUMB outperforms image replay methods such as iCARL by very large margins on all datasets (see Table I).

TABLE III

CRUMB Performs Better on Video Stream Learning With n Feature Map Representations in Its Replay Buffer (Ours) Than With n Raw Images (Im. Replay), Even Though the Former Uses Only 3.6% as Much Memory. This Finding Is Demonstrated for Both Class-iid and Class Instance, Across a Range of Buffer Sizes on CORe50, and Across Five Video Datasets, Although It Does Not Persist for Image Datasets Online-CIFAR100 and Online-ImageNet. For CORe50 in the "Dataset" Column, the Buffer Size Is Indicated as n_x. The Table Shows Mean Final Top-1 Accuracy on All Tasks, Averaged Across Five Independent Runs That Each Begins With an Independent Pretraining Run. Significant Differences From **Ours** Are Marked With *

Dataset	Experiment	Class-inst.	Class-iid
CORe50 ($n_x = 200$)	Ours	78.22	79.93
	Im. replay	67.80*	75.88*
CORe50 ($n_x = 400$)	Ours	79.08	81.37
	Im. replay	71.55*	78.36*
CORe50 ($n_x = 800$)	Ours	79.46	82.21
	Im. replay	68.73*	81.67
CORe50 ($n_x = 1600$)	Ours	79.30	82.60
	Im. replay	69.64*	79.62*
CORe50 ($n_x = 3200$)	Ours	81.39	80.83
	Im. replay	70.50*	79.72
CORe50 ($n_x = 6400$)	Ours	79.39	83.76
	Im. replay	69.60*	80.55*
Toybox	Ours	68.19	68.91
	Im. replay	64.03*	62.68*
iLab	Ours	67.80	71.96
	Im. replay	52.36*	63.38*
iCub	Ours	58.33	63.02
	Im. replay	50.50*	47.88*
iLab+CORe50	Ours	61.89	72.55
	Im. replay	59.05*	67.58*
Online-CIFAR100	Ours	-	47.97
	Im. replay	-	47.81
Online-ImageNet	Ours	-	23.99
	Im. replay	-	42.27*

Table III also shows that CRUMB continues to outperform image replay on CORe50 when memory resources for the replay buffer are not constrained. Even at $n_x = 6400$, where algorithms can store the entire CORe50 training set in the replay buffer, **Ours** outperforms **Image replay** by 9.8% on class instance and 3.2% on class-i.i.d.. The reduced performance of **Image replay** relative to **Ours** is partly rescued by adding CRUMB pretraining (**Ours p.t. + im. rep.**, 3.7% and 2.2% below **Ours** in class instance and class-i.i.d., respectively), even though the memory blocks play no role in either of the two image replay conditions during streaming (see Table IV).

Replaying high-level features also contributed to CRUMB's performance. Storing n_X low-level feature maps from layer 3 instead of layer 12 (**Early feature replay** versus **Ours**) reduced performance by 16.6%. CRUMB effectively stores memories with a higher level of abstraction than both **Image replay** and **Early feature replay** and comes with both accuracy and memory efficiency improvements. The memory recall (MeRec) method [48] uses a further level of abstraction by storing only the elementwise mean and standard deviation of feature activations for each learned class and generating examples for replay by sampling from a Gaussian distribution parameterized by these values. In **MeRec replay**, we implemented MeRec's replay mechanism as a drop-in replacement for CRUMB's memory block reconstruction at the same network layer and observed the accuracy of 56.9% and 41.3% below **Ours** for class instance and class-i.i.d., respectively, but 8.1% and 28.1% above **No replay**. MeRec stores an amount of data equivalent to two complete feature maps (mean and standard deviation) per class. In the implementation of CRUMB used for **Ours**, this is equivalent to 16 compressed feature maps per class or $n_x = 160$ in total, which is fewer than **Ours** at $n_x = 200$ but more than **Half capacity** at $n_x = 100$.

2) CRUMB Pretraining Primes CNN Weights for Streaming: Our model's performance is maximized by using CRUMB to pretrain the CNN and memory blocks on ImageNet prior to stream learning, particularly in the class-instance condition. Using randomly initialized memory blocks and a CNN pretrained without CRUMB (**Vanilla pretrain**) instead of CRUMB pretraining (**Ours**) reduced performance by 25.5% and 3.2% on class instance and class-i.i.d., respectively. Our results also indicate that the benefit of CRUMB pretraining is attributable primarily to changes in the CNN weights, rather than changes to the memory blocks. Starting stream learning with CRUMB-pretrained weights and randomly reinitialized memory blocks (**Pretrain weights**) performs only 1.1% and 0.5% worse than **Ours**, while starting with vanilla-pretrained weights and CRUMB-pretrained memory blocks (**Pretrain mem. blocks**) is 23.7% and 2.8% worse than **Ours**, a marginal improvement over vanilla pretraining. As explained in Section IV-D1, CRUMB pretraining also improves stream learning performance when raw images are used for replay.

CRUMB pretraining using the smaller CIFAR100 dataset (100 classes) instead of ImageNet (1000 classes) (**CIFAR100 pretrain**) decreases accuracy by 11.6%.

In **Freeze memory**, no updates to memory blocks were allowed after pretraining. This had no statistically significant effect on accuracy, indicating that fine-tuning the memory blocks was unnecessary for stream learning on CORe50.

3) CRUMB Pretraining Induces a Shape Bias in the CNN: We hypothesized that CRUMB's pretraining procedure induces a bias toward shape information over texture information by training the CNN to make class predictions using lossy feature representations with unperturbed spatial distributions. A bias toward shape information has been shown to help mitigate forgetting by flattening the local minima for each task in the loss landscape [72]. To gauge CRUMB's degree of reliance on shape and texture information, we evaluated its performance on test set examples with three different perturbations. "Spatial perturbation" and "feature perturbation" modify the $13 \times 13 \times 512$ feature map at the same level where it is reconstructed using memory blocks. In "spatial perturbation," the positions of all of the 512-D feature vectors are randomly shuffled in the 13×13 spatial grid, destroying global shape information but leaving feature information intact. In "feature perturbation," for each image independently, we set a random selection of 50% of the features (i.e., 256 out of 512 total features) to zero, perturbing feature information but leaving coarse shape information intact. Spatial and feature perturbations do not directly interact with CRUMB's reconstruction mechanism because CRUMB uses the original, nonreconstructed feature map to make class predictions. The "feature perturbation" strategy assumes that feature information is more related to texture information than shape information. Therefore, we also include "style perturbation" for ImageNet by testing on the Stylized-ImageNet dataset, which consists of images with heavily distorted local textures but largely intact global object shapes [71]. We would expect the performance of a relatively shape-biased network to be more severely reduced by spatial perturbation and less severely affected by feature and style perturbations than a control network. Fig. 4 shows CRUMB's "relative accuracy advantage" for each perturbation across several datasets. The relative accuracy advantage is calculated by dividing the accuracy drop (unperturbed accuracy minus perturbed accuracy) for each perturbation by the network's unperturbed accuracy and subtracting this value for CRUMB from the corresponding value for a control network. The "ImageNet (pretraining)" condition in Fig. 4 shows CRUMB's shape bias on ImageNet following pretraining on ImageNet, with lower resilience against spatial perturbations (red bars with diagonal lines) and higher

TABLE IV

ABLATION AND OTHER EXPERIMENTS DEMONSTRATE THE IMPORTANCE OF CRUMB'S VARIOUS COMPONENTS. TOP-1 ACCURACY ON ALL TASKS AFTER STREAM LEARNING IS AVERAGED OVER FIVE RUNS FOR ALL EXPERIMENTS. * DENOTES SIGNIFICANT DIFFERENCE FROM **OURS** ($p < 0.01$, PAIRED-SAMPLES t-TESTS ON BATCHES OF 100 IMAGES)

Category	Experiment name	Class-inst. % avg. accuracy	Class-iid % avg. accuracy
Unablated	**Ours**	**78.22**	**79.93**
Replay format	Image replay	67.80*	75.88*
	Ours p.t. + im. rep.	74.49*	77.72*
	Early feature replay	61.64*	64.28*
	MeRec replay [48]	21.35*	38.65*
Replay ablation	Half capacity	73.55*	75.14*
	Quarter capacity	64.90*	67.80*
	No replay	13.28*	10.58*
Pretraining ablation	Vanilla pretrain	52.70*	76.69*
	Pretrain weights	77.08*	79.40
	Pretrain mem. blocks	54.54*	77.18*
	CIFAR100 pretrain	66.64*	74.96*
Freeze memory	Freeze memory	78.06	80.44
Memory block init.	Normal init.	47.70*	74.28*
	Uniform init.	39.84*	64.82*
	Dense matched init.	77.24	78.84*
Number of memory blocks	1 block	9.60*	9.47*
	2 blocks	64.28*	71.23*
	4 blocks	70.10*	75.77*
	8 blocks	74.60*	79.96
	16 blocks	77.70	80.30
	...	...	...
	256 blocks (**Ours**)	**78.22**	**79.93**
	512 blocks	78.71	79.74
Memory block size	4-dim. blocks	74.21*	77.55*
	8-dim. blocks (**Ours**)	**78.22**	**79.93**
	16-dim. blocks	79.05*	81.02*
	32-dim. blocks	78.17	79.93
	16-dim. blocks adj.	79.69*	82.36*
	32-dim. blocks adj.	80.31*	81.64*
Loss functions	Ours - direct loss	73.82*	78.11*
	Ours + direct loss	65.40*	69.20*
	Direct loss	48.12*	50.07*
CNN Architecture	MobileNetV2 CNN	76.13*	79.27

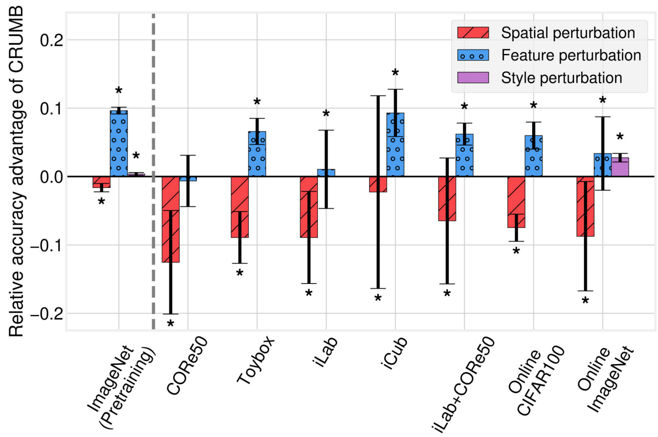


Fig. 4. CRUMB pretraining induces a bias toward shape information that often persists through stream learning. The height of each bar shows how much smaller (or larger if negative) CRUMB's drop in normalized test set accuracy under a perturbation is, in comparison to a control network (see Section IV-D2). "Spatial perturbation" shuffles the spatial positions of all feature vectors in an intermediate feature map (at the same layer where it is reconstructed by CRUMB), "feature perturbation" randomly sets half of the feature map's features to zero, and "style perturbation" uses images from Stylized-ImageNet [71]. Streaming results (to the right of gray dotted line) are in the class-instance setting for the video datasets and class-i.i.d. for CIFAR100 and ImageNet. Error bars are standard errors of the mean of relative accuracy advantage among five (CIFAR100 and ImageNet) or ten (other datasets) independent runs. * denotes a statistically significant difference from 0, as determined by a Wilcoxon signed-rank test (see Supplementary Section S4.B).

resilience against feature (blue with circles) and style (plain purple) perturbations. The control network is pretrained on ImageNet using the same procedure but without the CRUMB feature reconstruction step, thereby using only the "direct" loss for parameter updates. The other conditions in Fig. 4 correspond to CRUMB models evaluated on the test sets of these datasets after stream learning in class-instance (CORe50, Toybox, iLab, iCub, and iLab + CORe50) or class-i.i.d. (Online-CIFAR100 and Online-ImageNet) settings. Class-i.i.d. results for the video datasets are shown in Supplementary Fig. S3. Here, the control network performs stream learning with raw image replay and "direct" loss instead of CRUMB's feature-level replay and "codebook-out" loss. Although shape bias testing results are noisy on some of the datasets, CRUMB most often retains a degree of bias toward shape information over texture/feature information after the completion of stream learning.

4) 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 memory block count from **256 blocks** to as few as **16 blocks**, which effectively shrinks the library of feature combinations available to reconstruct feature maps, did not significantly decrease accuracy. Reducing the count further to **eight blocks** decreased accuracy by 3.6%, and reducing to **four** or **two blocks** decreased accuracy by 8.1% and 13.9%, respectively. Increasing to **512 blocks** did not significantly increase accuracy. This suggests a saturation effect, where a relatively small number of memory blocks are sufficient to reconstruct a wide variety of feature maps.

5) CRUMB Is Robust to Different Memory Block Sizes, and Memory 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, 4.0% and 2.4% on class instance and class-i.i.d., respectively. Increasing the number of elements from 8 to 16 or 32 (**16-dim. blocks** and **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 IV).

The maximum number of examples stored in CRUMB's replay buffer (n_x) was held constant for the memory block size perturbations above. However, increasing the dimension 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 image. This allows double/quadruple the number of examples to be stored in the replay buffer within the same memory budget. When we allowed the maximum number of examples stored in the buffer to change accordingly ($2n_x$ for **16-dim. blocks adj.** and $4n_x$ for **32-dim. blocks adj.**), we observed accuracy improvements: **16-dim. blocks adj.** achieves 1.5% and 2.4% higher accuracy than **Ours** (n_x with 8-D blocks) on class instance and class-i.i.d., respectively, and **32-dim. blocks adj.** achieves 2.1% and 1.7% higher accuracy. During hyperparameter tuning for our main results, we observed that 16-D memory blocks maximized testing accuracy.

6) Loss From Reconstructed Features Is Sufficient: CRUMB's performance is affected by the choice of components in its loss function. The loss function [see (3)] 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 4.4% drop in accuracy in the later stream learning tasks—learning from only reconstructed feature maps from start to finish, including during pretraining, is sufficient for decent performance. Including only codebook-out loss ("**Ours**") in stream learning yields a dramatic 30.1% gain in accuracy compared to using only direct loss ("**Direct loss**") and a gain of 12.8% compared to using a weighted sum of direct loss and codebook-out loss ("**Ours + direct loss**"), despite the fact that only the direct, nonreconstructed feature map is used for inference on the test set.

7) Initialization of the Memory Blocks Matters: CRUMB'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 the stream learning performance of four memory block initialization strategies, including initializing with values drawn from: 1) a standard normal distribution (**Normal init.**);

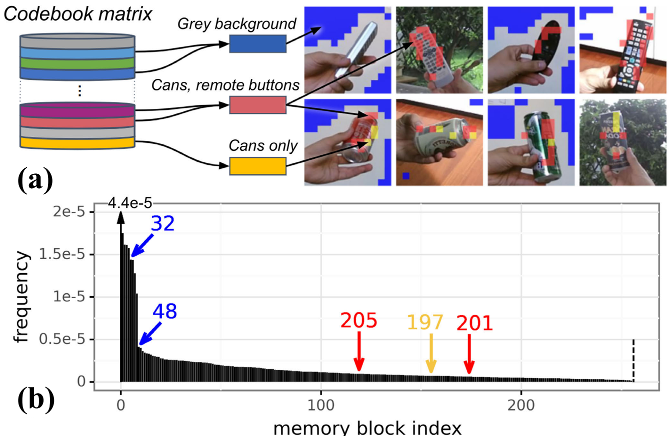


Fig. 5. Some memory blocks appear to have semantic interpretations. **a** Images of "remote controls" and "cans" in the CORe50 test set, showing all-or-none activation of specific memory blocks at the corresponding image locations. Of the 256 memory blocks in the codebook, blocks with indices 32 and 48 (blue squares) both similarly respond to grayish background regions, but not bright white or other backgrounds. Blocks 201 and 205 (red) both respond to buttons on remote controls and features of drink cans, while block 197 (yellow) responds only to can features. Similar blocks are aggregated by color (for blue and red) to produce a clearer visualization. (b) Sorted usage frequencies in the CORe50 test set of each of the 256 memory blocks. Colored arrows show the blocks visualized in (a). The upward black arrow shows the most-used block with frequency 4.4e^{-5}.

2) a uniform distribution on the interval $[0, 1]$ (**Uniform init.**); 3) a distribution designed to match that of the nonzero 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**); and 4) the same as 3), but with no values set to zero (**Dense matched init.**). Accuracy for **Normal init.** was 30.5% and 5.7% lower than **Ours** for class-instance and class-i.i.d. protocols, respectively, accuracy for **Uniform init.** was 38.4% and 15.1% lower, and accuracy for **Dense matched init.** was 1.0% ($p = 0.045 > 0.01$) and 1.1% lower (see Table IV). It appears that drawing initial values for the memory blocks from a similar distribution to that of natural feature maps improves the 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 sampling a value from this vector.

8) CRUMB Is Applicable Across CNN Architectures: In **MobileNetV2 CNN**, we implement CRUMB using the MobileNetV2 [73] CNN backbone instead of SqueezeNet [59]. Here, CRUMB reconstructs the $14 \times 14 \times 64$ input to MobileNetV2's sixth layer, using 256 8-D memory blocks as in **Ours**. With the total memory usage of the replay buffer held constant, the performance of **MobileNetV2 CNN** is comparable to **Ours** with only a 2.1% accuracy drop in class instance and a 0.7% (not significant) drop in class-i.i.d.

9) Some Memory Blocks Are Coarsely Interpretable: Visualizations of image locations where specific memory blocks are activated (Fig. 5) 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 of classes, and others responded to features that are likely irrelevant to classification. In addition to the blocks visualized in Fig. 5, 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. Given the observations earlier in this section that either randomly reinitializing memory blocks prior to stream learning (**Pretrain weights**) or freezing memory blocks during stream learning (**Freeze memory**) has minimal effects on performance, it is not necessarily the case that these interpretable associations indicate learned representations within the memory blocks themselves. Another possibility is that a sufficient diversity of memory blocks allows useful associations between memory blocks and features or classes to be learned by the CNN through changes to the network weights.

The procedure for generating the visualizations in Fig. 5(a) can be understood as follows. For this analysis, we used a CRUMB model trained on CORe50 in the class-instance setting. CORe50 test set images are first passed through the early layers of the CNN 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 Section III-B2). In this study, each feature map is of size $13 \times 13 \times 512$, meaning spatial dimensions of 13×13 with 512 features at each spatial location. Each memory block is one of the 256 row vectors in the 256×8 codebook matrix used for this analysis. The memory blocks are 8-D vectors, so each spatial location in the feature map's 13×13 grid is represented by a 512-D vector formed by concatenating $512/8 = 64$ memory blocks end-to-end. Color coding in Fig. 5(a) shows at most one block per spatial location, the one activated by the first eight features in the 512-D feature vector, even though 64 memory blocks are activated at each location in total. We focus on the first eight features for visualization purposes because it is not necessarily the case that blocks activated by the first set of eight features encode the same image features as they might when activated by the kth set of eight features (where $2 \leq k \leq 64$). To produce the images in Fig. 5(a), each test set image is divided into a square 13×13 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 eight 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 the same color to sets of memory blocks that seemed to respond to very similar features. Fig. 5(b) shows the sorted distribution of the frequencies with which each of the 256 memory blocks was used to reconstruct feature maps from the CORe50 test set, with color and memory block index-coded arrows indicating the memory blocks visualized in Fig. 5(a).

V. Conclusion and Discussion

We developed a novel compositional replay strategy to tackle the problem of online stream learning, in which algorithms must learn tasks incrementally from nonrepeating, temporally correlated inputs. Our algorithm, CRUMB, learns a set of "memory blocks" that are selected via cosine similarity and concatenated to reconstruct feature maps from an intermediate CNN layer. 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 state-of-the-art online stream learning accuracy across seven datasets. Furthermore, CRUMB outperforms replay of an equal number of raw images by large accuracy margins across five video datasets, despite using only 3.6% as much memory as image replay.

Several factors seem to make important contributions to CRUMB's high performance. As shown in Fig. 4, pretraining with memory block reconstruction biases the CNN toward attending to object shapes rather than textures, an effect that typically endures throughout stream learning and which has been demonstrated to reduce catastrophic forgetting by flattening the loss minima for each task [72]. This could explain why CRUMB pretraining improves performance even if raw image replay is used during stream learning ("Ours p.t. + im. rep." in Table IV), in which case feature map reconstruction plays no role whatsoever during stream learning.

Backpropagation updates to memory blocks during pretraining and stream learning appear to be less important for performance than updates to the CNN weights. Randomly reinitializing the memory blocks after pretraining has a small negative effect on performance only in the class-instance setting while keeping only the pretrained memory blocks but resetting to the original "vanilla" pretrained CNN weights before stream learning has a much larger negative impact ("pretrain weights" versus "pretrain mem. blocks" in Table IV). Furthermore, freezing the memory blocks during streaming has no discernible effect ("freeze memory" in Table IV). However, "normal init." and "uniform init." in Table IV show that the choice of probability distribution used to randomly initialize the memory blocks can have a dramatic impact on performance, with best performance attained by matching the univariate distribution of the memory blocks to that of natural feature maps. It seems to be important for stream learning that the later layers of the network receive feature maps with consistent univariate statistics, whether they are natural feature maps from the feature extractor layers or reconstructed feature maps from memory block concatenation.

Experiments in Table IV also 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 natural feature maps, either together with codebook-out loss ("ours + direct loss") or in isolation ("direct loss"). Only codebook-out loss is available when replaying feature maps reconstructed from memory blocks: using only codebook-out loss for new images means that only codebook-out loss is used throughout stream learning, rather than switching between direct loss for new examples and codebook-out loss for replayed examples. It appears that CRUMB's memory blocks form a shared, discretized basis for encoding training examples in feature space, which has a stabilizing effect on CNN during stream learning. It is also notable in this context that, unlike during streaming, the inclusion of direct loss is important for gradient updates during pretraining ("Ours − direct loss" has lower accuracy than "Ours" in Table IV). Combined with the observation that memory blocks should ideally be initialized from a univariate distribution approximating that of natural feature maps, this suggests that the pretraining process helps the CNN align its processing of natural and reconstructed feature maps, enabling the network to learn only from reconstructed feature maps during streaming even while continuing to make its most accurate predictions using natural feature maps instead.

The hypothesis that CRUMB's memory blocks provide a shared feature-level basis that stabilizes the CNN is consistent with the observation that, although CRUMB pretraining improves the performance of raw image replay ("Ours p.t. + im. rep." in Table IV), it still does not match CRUMB's performance with the replay buffer size n_x held constant: we speculate that redundant pixel-level information in raw images introduces additional noisy variation into the distribution of feature maps, which affects network stability. Another observation consistent with this hypothesis is that CRUMB only outperforms raw image replay on the five video datasets (with constant n_x, not constant memory usage), where network instability is likely to be more problematic due to temporal correlations within video clips and sudden transitions between them during training.

In both CRUMB and our raw image replay ablation experiments, the early "feature extractor" layers of the CNN are frozen. When we apply CRUMB reconstruction to feature maps from an earlier layer ("early feature replay" in Table IV) and correspondingly allow more layers after this point to have their parameters updated during stream learning, we observe performance worse than both CRUMB and image replay. One interpretation of this is that more unfrozen layers means that more network parameters are exposed to gradient updates and, consequently, to catastrophic forgetting. It is also possible that CRUMB's reconstruction mechanism is best suited to representing abstract, high-level features that are more likely to be found in later CNN layers, and that too much information is lost if CRUMB attempts to represent lower level information that is interpreted by later layers in more granular ways.

Given the apparent necessity of preserving the information in feature maps during reconstruction, a surprisingly small codebook of memory blocks is sufficient. As few as 16 memory blocks are needed for optimal performance on CORe50, CRUMB still performs remarkably well with only two memory blocks (see "number of memory blocks" experiments in Table IV). Although CRUMB is already highly memory-efficient with the memory blocks themselves occupying negligible space, reducing the number of memory blocks may enable further CPU memory usage optimizations (e.g., 4-bit integers as indices for 16 memory blocks) and also lowers GPU memory usage. Computational and memory efficiency is presumably critical in biological memory systems. Indeed, replay of neuronal activity patterns has been observed to help reinforce and consolidate memories in multiple brain areas across different mammalian species [50], [51], [52]. It is unlikely that neural circuits in the brain use replay mechanisms that preserve as much low-level information as pixel-level replay. Instead, it is interesting to speculate that one of the mechanisms by which brains avoid catastrophic forgetting is by replaying compositions of abstract, high-level features in a manner analogous to CRUMB's replay mechanism.

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 could also be used in federated learning contexts, enabling highly effective replay of previously seen data points via perhaps unrecognizably lossy representations, thereby minimizing both catastrophic forgetting and data security risks. The interpretable qualities of a subset of memory blocks, however, raise the possibility of identifying weak associations with certain generic features contained in a given training example, for a person with unauthorized access to CNN weights, memory blocks, encoded memories of interest, and a reference dataset to discover memory block interpretations. However, it would still be impossible for such a person to completely reconstruct CRUMB's memories in their original encodings (e.g., in pixels).

CRUMB is implemented here for SqueezeNet [59] and MobileNetV2 [73] CNNs but could be used to mitigate forgetting across different neural network architectures in the future. For example, memory blocks could be used to efficiently reconstruct and replay vector outputs of self-attention heads at intermediate layers in transformer models [74], [75].

Updating CRUMB's memory blocks using backpropagation in tandem with network weights is highly 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, it is possible that fine-tuning may be necessary for tasks with substantial nonstationarity. In addition, in this study, CRUMB does not adapt the early "feature extractor" layers of the CNN during stream learning. However, the early layers could theoretically be trained using the direct prediction loss while the late layers and memory blocks are trained using codebook-out loss or a combination of these two losses: this approach could enable additional flexibility for domain adaptation. Future studies could apply CRUMB to stream learning or reinforcement learning tasks with shifting domains, emulating humans or robots in continuously changing environments.

Acknowledgment

The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institute of General Medical Sciences or the National Institutes of Health. The authors would like to thank Trenton Bricken, Spandan Madan, and Warren Sunada-Wong for sharing their helpful insights into their work at various times during this project.

References

[1] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of Learning and Motivation*, vol. 24. Amsterdam, The Netherlands: Elsevier, 1989, pp. 109–165.

[2] R. Ratcliff, "Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions," *Psychol. Rev.*, vol. 97, no. 2, pp. 285–308, 1990.

[3] R. French, "Catastrophic forgetting in connectionist networks," *Trends Cognit. Sci.*, vol. 3, no. 4, pp. 128–135, Apr. 1999.

[4] T. L. Hayes, K. Kafle, R. Shrestha, M. Acharya, and C. Kanan, "Remind your neural network to prevent catastrophic forgetting," in *Proc. Eur. Conf. Comput. Vis.* Heidelberg, Germany: Springer, 2020, pp. 466–483.

[5] R. Kemker, M. McClure, A. Abitino, T. L. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," in *Proc. 32nd AAAI Conf. Artif. Intell.*, vol. 32, no. 1, 2018.

[6] D. Maltoni and V. Lomonaco, "Continuous learning in single-incremental-task scenarios," *Neural Netw.*, vol. 116, pp. 56–73, Aug. 2019.

[7] M. De Lange et al., "A continual learning survey: Defying forgetting in classification tasks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3366–3385, Jul. 2022.

[8] I. Evron, E. Moroshko, R. Ward, N. Srebro, and D. Soudry, "How catastrophic can catastrophic forgetting be in linear regression?" in *Proc. Conf. Learn. Theory*, 2022, pp. 4028–4079.

[9] H. Vaidya, T. Desell, and A. G. Ororbia, "Reducing catastrophic forgetting in self organizing maps with internally-induced generative replay (student abstract)," in *Proc. AAAI Conf. Artif. Intell.*, 2022, vol. 36, no. 11, pp. 13069–13070.

[10] C. Shao and Y. Feng, "Overcoming catastrophic forgetting beyond continual learning: Balanced training for neural machine translation," in *Proc. 60th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1, 2022, pp. 2023–2036.

[11] A. Prabhu, P. H. Torr, and P. K. Dokania, "GDumb: A simple approach that questions our progress in continual learning," in *Proc. Eur. Conf. Comput. Vis.* Heidelberg, Germany: Springer, 2020, pp. 524–540.

[12] K. James et al., "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 13, pp. 3521–3526, Mar. 2017.

[13] C. S. Lee and A. Y. Lee, "Clinical applications of continual learning machine learning," *Lancet Digit. Health*, vol. 2, no. 6, pp. e279–e281, Jun. 2020.

[14] X. Wang et al., "The toybox dataset of egocentric visual object transformations," 2018, *arXiv:1806.06034*.

[15] A. Borji, S. Izadi, and L. Itti, "ILab-20M: A large-scale controlled object dataset to investigate deep learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2221–2230.

[16] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2935–2947, Dec. 2018.

[17] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with A-GEM," 2018, *arXiv:1812.00420*.

[18] X. He and H. Jaeger, "Overcoming catastrophic interference using conceptor-aided backpropagation," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, Vancouver, BC, Canada, Apr./May 2018.

[19] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 3987–3995.

[20] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4652–4662.

[21] Y. Kong, L. Liu, H. Chen, J. Kacprzyk, and D. Tao, "Overcoming catastrophic forgetting in continual learning by exploring eigenvalues of Hessian matrix," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–15, Jul. 2023.

[22] J. Wen, Y. Cao, and R. Huang, "Few-shot self reminder to overcome catastrophic forgetting," 2018, *arXiv:1812.00543*.

[23] Z. Mai, R. Li, J. Jeong, D. Quispe, H. Kim, and S. Sanner, "Online continual learning in image classification: An empirical survey," *Neurocomputing*, vol. 469, pp. 28–51, Jan. 2022.

[24] Z. Zhang, Y. Chen, and C. Zhou, "Self-growing binary activation network: A novel deep learning model with dynamic architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–10, May 2022.

[25] C. Fernando et al., “PathNet: Evolution channels gradient descent in super neural networks,” 2017, *arXiv:1701.08734*.

[26] J. Rajasegaran, M. Hayat, S. H. Khan, F. S. Khan, and L. Shao, “Random path selection for continual learning,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32. Red Hook, NY, USA: Curran Associates, 2019.

[27] J. Serra, D. Suris, M. Miron, and A. Karatzoglou, “Overcoming catastrophic forgetting with hard attention to the task,” in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4548–4557.

[28] T. Adel, H. Zhao, and R. E. Turner, “Continual learning with adaptive weights (CLAW),” 2019, *arXiv:1911.09514*.

[29] T. Bricken, X. Davies, D. Singh, D. Krotov, and G. Kreiman, “Sparse distributed memory is a continual learner,” in *Proc. 11th Int. Conf. Learn. Represent. (ICLR)*, Kigali, Rwanda, May 2023.

[30] J. Schwarz et al., “Progress & compress: A scalable framework for continual learning,” 2018, *arXiv:1805.06370*.

[31] S. Golkar, M. Kagan, and K. Cho, “Continual learning via neural pruning,” 2019, *arXiv:1903.04476*.

[32] J. Peng et al., “Overcoming long-term catastrophic forgetting through adversarial neural pruning and synaptic consolidation,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 9, pp. 4243–4256, Sep. 2022.

[33] Q. Gao, Z. Luo, D. Klabjan, and F. Zhang, “Efficient architecture search for continual learning,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 11, pp. 1–11, Nov. 2023.

[34] Y. Wu et al., “Large scale incremental learning,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 374–382.

[35] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, “ICaRL: Incremental classifier and representation learning,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5533–5542.

[36] R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio, “Gradient based sample selection for online continual learning,” 2019, *arXiv:1903.08671*.

[37] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner, “Variational continual learning,” 2017, *arXiv:1710.10628*.

[38] D. Lopez-Paz et al., “Gradient episodic memory for continual learning,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6467–6476.

[39] J. Bang, H. Kim, Y. Yoo, J.-W. Ha, and J. Choi, “Rainbow memory: Continual learning with a memory of diverse samples,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 8214–8223.

[40] H. Shin, J. K. Lee, J. Kim, and J. Kim, “Continual learning with deep generative replay,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 2990–2999.

[41] A. Robins, “Catastrophic forgetting, rehearsal and pseudorehearsal,” *Connection Sci.*, vol. 7, no. 2, pp. 123–146, Jun. 1995.

[42] Y. Liu, Y. Su, A.-A. Liu, B. Schiele, and Q. Sun, “Mnemonics training: Multi-class incremental learning without forgetting,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 12242–12251.

[43] C. Atkinson, B. McCane, L. Szymanski, and A. Robins, “Pseudo-recursal: Solving the catastrophic forgetting problem in deep neural networks,” 2018, *arXiv:1802.03875*.

[44] X. Liu et al., “Generative feature replay for class-incremental learning,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2020, pp. 915–924.

[45] G. Shen, S. Zhang, X. Chen, and Z.-H. Deng, “Generative feature replay with orthogonal weight modification for continual learning,” in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–8.

[46] G. M. van de Ven, Z. Li, and A. S. Tolias, “Class-incremental learning with generative classifiers,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2021, pp. 3606–3615.

[47] L. Pellegrini, G. Graffieti, V. Lomonaco, and D. Maltoni, “Latent replay for real-time continual learning,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 10203–10209.

[48] B. Zhang, Y. Guo, Y. Li, Y. He, H. Wang, and Q. Dai, “Memory recall: A simple neural network training framework against catastrophic forgetting,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 5, pp. 2010–2022, May 2022.

[49] H. Jégou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2011.

[50] J. O’Neill, B. Pleydell-Bouverie, D. Dupret, and J. Csicsvari, “Play it again: Reactivation of waking experience and memory,” *Trends Neurosciences*, vol. 33, no. 5, pp. 220–229, May 2010.

[51] P. A. Lewis and S. J. Durrant, “Overlapping memory replay during sleep builds cognitive schemata,” *Trends Cognit. Sci.*, vol. 15, no. 8, pp. 343–351, Aug. 2011.

[52] J.-B. Eichenlaub et al., “Replay of learned neural firing sequences during rest in human motor cortex,” *Cell Rep.*, vol. 31, no. 5, May 2020, Art. no. 107581.

[53] J. L. McClelland, B. L. McNaughton, and R. C. O’Reilly, “Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory,” *Psychol. Rev.*, vol. 102, no. 3, p. 419, 1995.

[54] J. Peng, D. Ye, B. Tang, Y. Lei, Y. Liu, and H. Li, “Lifelong learning with cycle memory networks,” *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–14, Jul. 2023.

[55] V. Lomonaco and D. Maltoni, “Core50: A new dataset and benchmark for continuous object recognition,” in *Proc. Conf. Robot Learn.*, 2017, pp. 17–26.

[56] G. Pasquale, C. Ciliberto, L. Rosasco, and L. Natale, “Object identification from few examples by improving the invariance of a deep convolutional neural network,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2016, pp. 4904–4911.

[57] A. Krizhevsky et al., “Learning multiple layers of features from tiny images,” M.S thesis, Univ. Toronto, Toronto, ON, Canada, 2009.

[58] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[59] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5MB model size,” 2016, *arXiv:1602.07360*.

[60] Y. Liu, B. Schiele, and Q. Sun, “Adaptive aggregation networks for class-incremental learning,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 2544–2553.

[61] M. De Lange and T. Tuytelaars, “Continual prototype evolution: Learning online from non-stationary data streams,” 2020, *arXiv:2009.00919*.

[62] S. I. Mirzadeh, M. Farajtabar, R. Pascanu, and H. Ghasemzadeh, “Understanding the role of training regimes in continual learning,” 2020, *arXiv:2006.06958*.

[63] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[64] A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library,” 2019, *arXiv:1912.01703*.

[65] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, “Memory aware synapses: Learning what (not) to forget,” in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 139–154.

[66] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?” 2014, *arXiv:1411.1792*.

[67] Y. Chen, M. Welling, and A. Smola, “Super-samples from kernel herding,” 2012, *arXiv:1203.3472*.

[68] P. W. Koh and P. Liang, “Understanding black-box predictions via influence functions,” in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 1885–1894.

[69] P. P. Brahma and A. Othon, “Subset replay based continual learning for scalable improvement of autonomous systems,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2018, pp. 1179–11798.

[70] S. Ho, M. Liu, L. Du, L. Gao, and Y. Xiang, “Prototype-guided memory replay for continual learning,” *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–11, Mar. 2023.

[71] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, “ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness,” in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, New Orleans, LA, USA, May 2019.

[72] Z. Shi, Y. Sun, J. Hwee Lim, and M. Zhang, “On the robustness, generalization, and forgetting of shape-texture debiased continual learning,” 2022, *arXiv:2211.11174*.

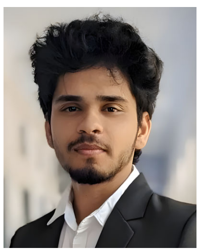
[73] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[74] A. Vaswani et al., “Attention is all you need,” 2017, *arXiv:1706.03762*.

[75] A. Dosovitskiy et al., "An image is worth 16×16 words: Transformers for image recognition at scale," in *Proc. 9th Int. Conf. Learn. Represent. (ICLR)*, May 2021.

Morgan B. Talbot is currently pursuing the M.D. and Ph.D. degrees with the Harvard-MIT Program in Health Sciences and Technology.

His research interests include learning and memory in humans and machines, as well as applicationsa of artificial intelligence to psychiatry and mental health.

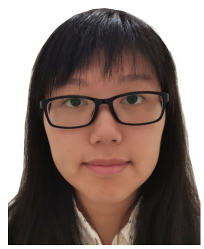


Rushikesh Zawar received the B.E. degree in computer science and the M.Sc. degree in biological sciences from the Birla Institute of Technology and Science at Pilani (BITS Pilani), Pilani, India. He completed his thesis at Harvard University, Cambridge, MA, USA.

His research interests include computer vision, reinforcement learning, neuroscience, and the intersection of artificial intelligence and biology.

Rohil Badkundri is currently pursuing the bachelor's degree in applied math with Harvard University, Cambridge, MA, USA.

He is broadly interested in applications at the intersection of machine learning and biology.



Mengmi Zhang is currently an Assistant Professor and a Principal Investigator of the Deep NeuroCognition Laboratory, Nanyang Technological University (NTU), Singapore, and the Agency for Science, Technology and Research (A*STAR), Singapore. Her research background is at the intersection of artificial intelligence and computational neuroscience. She has made contributions to understanding gaze anticipation, zero-shot visual search, contextual reasoning, and continual learning.

Gabriel Kreiman is currently a Professor at the Harvard Medical School, Boston, MA, USA, and Boston Children's Hospital, Boston, and leads the Executive Function/Memory Module at the Center for Brains, Minds and Machines. His research group combines computational models, behavioral measurements, and neurophysiological recordings to study visual cognition, learning, and memory.