Discrete Key-Value Bottleneck

Frederik Träuble [1] **Anirudh Goyal** [2] **Nasim Rahaman** [1 3] **Michael Mozer** [4]
Kenji Kawaguchi [5] **Yoshua Bengio** [3 6 7] **Bernhard Schölkopf** [1 7]

Abstract

Deep neural networks perform well on classification tasks where data streams are i.i.d. and labeled data is abundant. Challenges emerge with non-stationary training data streams such as continual learning. One powerful approach that has addressed this challenge involves pre-training of large encoders on volumes of readily available data, followed by task-specific tuning. Given a new task, however, updating the weights of these encoders is challenging as a large number of weights needs to be fine-tuned, and as a result, they forget information about the previous tasks. In the present work, we propose a model architecture to address this issue, building upon a discrete bottleneck containing pairs of separate and learnable key-value codes. Our paradigm will be to *encode; process the representation via a discrete bottleneck; and decode*. Here, the input is fed to the pre-trained encoder, the output of the encoder is used to select the nearest keys, and the corresponding values are fed to the decoder to solve the current task. The model can only fetch and re-use a sparse number of these key-value pairs during inference, enabling *localized and context-dependent model updates*. We theoretically investigate the ability of the discrete key-value bottleneck to minimize the effect of learning under distribution shifts and show that it reduces the complexity of the hypothesis class. We empirically verify the proposed method under challenging class-incremental learning scenarios and show that the proposed model — without any task boundaries — reduces catastrophic forgetting across a wide variety of pre-trained models, outperforming relevant baselines on this task.

[1]MPI for Intelligent Systems, Tübingen [2]Google DeepMind [3]Mila [4]Google Research, Brain Team [5]National University of Singapore [6]Université de Montréal [7]CIFAR Fellow. Correspondence to: Frederik Träuble <frederik.traeuble@tuebingen.mpg.de>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

1. Introduction

Current neural networks achieve state-of-the-art results across a range of challenging problem domains. Most of these advances are limited to learning scenarios where the training data is sampled i.i.d. from some assumed joint distribution $P(X, Y) = P(Y|X)P(X)$ in the supervised case. In reality, learning scenarios are often far from i.i.d. and exhibit various distribution shifts; for instance during continual learning, when the training distribution changes over time.

Despite all successes, the training of deep neural networks on non-stationary training data streams remains challenging. For instance, training neural networks without additional modifications on some small dataset or input distribution may quickly lead to over-fitting and catastrophic forgetting of previous knowledge (Chen & Liu, 2018; Thrun, 1995; Van de Ven & Tolias, 2019). One paradigm that has dominated in recent work is the use of models pre-trained on large amounts of data e.g. in a self-supervised manner (Chen et al., 2020; Azizi et al., 2021; Caron et al., 2021; Brown et al., 2020). These models are then re-purposed for smaller datasets via either (a) fine-tuning, which may involve changing large numbers of parameters, or (b) introducing a small number of new parameters to adapt the model (Houlsby et al., 2019; Zhou et al., 2022b; Wang et al., 2022).

Building upon this pre-training paradigm, we introduce an approach (see Figure 1 for an overview of the model architecture) to distil information into a discrete set of code pairs, each consisting of a coupled key and value code. In a nutshell, we follow a three-step process: *encode, process via a discrete bottleneck, and decode*.

In the first step, an input is fed to the encoder to obtain a continuous-valued representation that is projected into C lower-dimensional heads. In the second step, these heads are used to search for the closest key within head-specific key-value codebooks, and *corresponding* continuously learnable value codes taken from a discrete set are fetched. In the third step, these value codes are passed through a downstream decoder to produce the final output. By freezing all components except for the value codes, the model can continue to improve its performance *locally* on input-dependent value codes without affecting the prediction and key-value pairs of prior training samples.

As we will show theoretically and empirically, this architecture offers an appealing ability to improve generalization under input distribution shifts (also called *covariate shifts*) without common vulnerabilities to non-i.i.d. training distributions. We will show that we can mitigate forgetting via localization because only fetched value codes are being updated. Theoretically (see Section 3), this is enabled via the following two mechanisms: first, having these intermediate discrete codes minimizes the nonstationarity of the input distribution shift by keeping the joint features fixed; and second, the discrete bottleneck will reduce the complexity of the hypothesis class through its separate key-value codes.

The fundamental difference to prior work on discrete bottlenecks (Van Den Oord et al., 2017; Razavi et al., 2019; Liu et al., 2021) is the introduction of the discrete pairs with two different codes, the first (key codes) being optimized for the encoder and the second (value codes) being optimized for the decoder. This means that the gradients from a particular downstream task will not directly affect the key codes, which induces a rather different learning behaviour compared to standard neural networks.

Contributions: We introduce a model architecture designed for learning under input distribution changes during training. We corroborate this theoretically by proving that under input distribution shifts, the proposed model achieves an architecture-dependent generalization bound that is better than that of models without the discrete bottleneck. We empirically verified the method under a challenging class-incremental learning scenario on real-world data and show that the proposed model — without any task boundaries — reduces the common vulnerability to catastrophic forgetting across a wide variety of publicly available pre-trained models and outperforms relevant baselines on this task.

2. Encode, Processing via Discrete Key-Value Bottleneck, Decode

This section introduces the proposed model architecture. An overview of the workings of the model is given in Figure 1. Our goal is to learn a model $f_\theta : \mathcal{X} \to \mathcal{Y}$ from training data $S = ((x_i, y_i))_{i=1}^n$ that is robust to strong input distribution changes. We consider a model that can be decomposed as

$$f_\theta(x) = (d_\delta \circ v_\gamma \circ k_\beta \circ z_\alpha)(x) \tag{1}$$

with $\theta = (\alpha, \beta, \gamma, \delta)$. In the first step, the encoder $z_\alpha : \mathcal{X} \to \mathcal{Z} \in \mathbb{R}^{m_z}$ extracts a representation from the high-dimensional observation x. We further project this representation into C lower-dimensional feature heads, each of them being passed as the input into a separate head-specific learnable key-value codebook. As the name suggests, a key-value codebook is a bijection that maps each key vector to its value (where the latter is learned). Within each codebook, a quantization process k_β selects the closest key to its input head (think of it as a "feature anchor") from a head-specific key-value codebook. Next, a corresponding value vector is being fetched and returned (one per codebook), which we denote via v_γ. The returned values across all codebooks finally serve as input to a decoder d_δ for prediction. In the following, each component is motivated in detail.

Step 1: Encoding. The encoder z_α projects the input x into a lower-dimensional vector $z \in \mathbb{R}^{m_z}$, further down-projected into C separate heads of dimension d_{key} using C fixed Gaussian random projection matrices. For tasks where we judge x to be sufficiently low-dimensional, we skip encoding and partition x directly. For the purpose of this work we take a broad spectrum of differently pre-trained encoders (Caron et al., 2021; 2020; He et al., 2016; Trockman & Kolter, 2022; Radford et al., 2021; Dosovitskiy et al., 2020), and show that they are all capable of embedding features that are relevant for the studied task.

Step 2: Processing via Discrete Key-Value Bottleneck.

Step 2.1: Quantization. A quantization process snaps each head to the closest key from the corresponding head's codebook, based on the head's content. Closeness is defined by the L2 distance between the head and key vectors.

Step 2.2: Retrieval. For each head, a simple lookup in the head-specific codebook is performed to retrieve the value corresponding to the key selected in the previous step.

Overall, the bottleneck will retrieve a set of C value codes.

Step 3: Decoding. Finally, we predict the target variable from this set of fetched values using any suitable decoder function d_δ. For the purpose of our experiments on classification, we can apply a simple non-parametric decoder function which uses average pooling to calculate the element-wise average of all the fetched value codes and then applies a softmax function to the output.

Initializing the Model. During training, we first initialize the key codes. The non-parametric 1-to-1 mapping between key and value codes means no gradient back-propagates from the values to the keys. Instead, they are solely determined from the encoding of the input observation, implying that no supervised data is required to learn them. A simple and effective way to initialize the keys is via exponential moving average (EMA) updates as proposed for VQ-VAE in (Van Den Oord et al., 2017) and used for VQ-VAE2 in (Razavi et al., 2019). This procedure allows us to obtain keys that are broadly distributed in the feature space of the encoder, without introducing additional parameters. We refer to Appendices C and E.3 and Section 5.3 for more details on the implementation and robustness of this procedure. Once initialized, the keys are frozen and thus not influenced by the task on the target domain, implying that any given input x is mapped to keys from a fixed set.

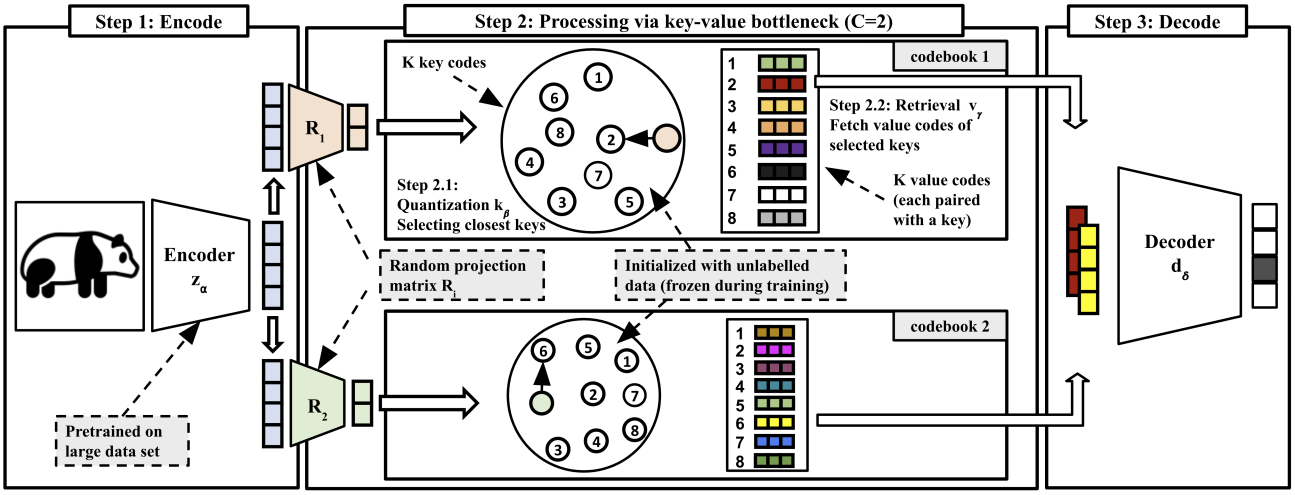

Figure 1. **Overview of discrete key-value bottleneck**. First, a pre-trained encoder maps the high-dimensional input to a lower-dimensional representation. Second, this representation is projected into C lower-dimensional heads. $C = 2$ is chosen for illustrative purposes. Each head is processed as input by one out of C separate key-value codebooks: In the codebook, the head is discretized by snapping to the closest key code, and the corresponding value code for that key is fetched. Third, the values from different key-value codebooks are combined and fed as input to the decoder for the final prediction. Keys and values can be of different dimensions in the proposed model.

Training under Dataset Shifts. Assume we are now given training data $S = ((x_i, y_i))_{i=1}^n$, but the data generation process might undergo various covariate shifts of $P(X)$ without knowing when they occur. A particularly challenging example of that would be the task of continual learning class-incrementally without having task boundaries, where the support of $P(X)$ changes dramatically. Having the encoder and decoder solely connected through the key-value codes, we can now train the model on this data stream by only updating the value codes under such an input distribution shift. Since we only locally update the actual retrieved values under the input distribution changes, the values from other data domains remain unchanged, thereby enabling *localized, context-dependent model updates*. In this way, we can integrate new data and thus gradually improve the models' performance. Moreover, as we will show theoretically, the proposed architecture benefits from the fact that the decoder works with a discrete set of value codes, as opposed to directly predicting from the encoder representation.

3. Theoretical Analysis

In this section, we theoretically investigate the behavior of the proposed method. In contrast to standard models, the proposed model is shown to have the ability to reduce the generalization error of transfer learning under input distribution shifts via two mechanisms: (1) the discretization with key-value pairs, mitigating the effect of the input distribution shift, and (2) the discrete bottleneck with key-value pairs, reducing the complexity of the hypothesis class.

To formally state this, we first introduce the notation. We consider the distribution shift of x via an arbitrary function $(x, \epsilon) \mapsto g_\epsilon(x)$ with random vectors ϵ. In the target domain, we are given a training dataset $S = ((x_i, y_i))_{i=1}^n$ of n samples where $x_i \in \mathcal{X} \subseteq \mathbb{R}^{m_x}$ and $y_i \in \mathcal{Y} \subseteq \mathbb{R}^{m_y}$ are the i-th input and the i-th target respectively. The training loss per sample is defined by $\ell(f_\theta(x_i), y_i)$, where $\ell : \mathbb{R}^{m_y} \times \mathcal{Y} \to [0, M]$ is a bounded loss function. We compare the proposed model against an arbitrary model $\hat{f} : \mathcal{X} \to \mathbb{R}^{m_y}$ without the key-value discretization with a hypothesis class $\hat{\mathcal{F}} \ni \hat{f}$. Let f_θ^S (for the proposed model) and $\hat{f}^S$ (for an arbitrary model) be the corresponding hypotheses learned through the training set S. Let $\mathcal{K} = \{(k_\beta \circ z_\alpha)(x) : x \in \mathcal{X}\}$ be the set of keys. By ordering elements of $\mathcal{K}$, we use $\mathcal{K}_i$ to denote i-th element of $\mathcal{K}$. Let $\hat{d}$ be a distance function. We define $[n] = \{1, \ldots, n\}$, $\mathcal{C}_k = \{x \in \mathcal{X} : k = \operatorname{argmin}_{i \in [|\mathcal{K}|]} \hat{d}((k_\beta \circ z_\alpha)(x), \mathcal{K}_i)\}$ (the input region corresponding to the k-th key), and $\mathcal{I}_k^y = \{i \in [n] : x_i \in \mathcal{C}_k, y_i = y\}$ (the data indices corresponding to the k-th key and the label y). We define $\mathbb{1}\{a = b\} = 1$ if $a = b$ and $\mathbb{1}\{a = b\} = 0$ if $a \neq b$. We denote by $\mathcal{D}_{x|y}$ the conditional distribution of the input x given a label y, and by $\bar{S} = ((\bar{x}_i, \bar{y}_i))_{i=1}^n$ the random variable having the same distribution as S.

The following theorem (proof in Appendix A) shows that under input distribution shifts, a model with the proposed architecture has the ability to reduce the generalization error of the base model:

Theorem 3.1. *There exists a constant c (independent of $n, f, \hat{\mathcal{F}}, \delta, \epsilon$ and S) such that for any $\delta > 0$, with probability at least $1-\delta$, the following holds for any $f \in \{f_\theta^S, \hat{f}^S\}$:*

$$\mathbb{E}_{(x,y),\epsilon}[\ell(f(g_\epsilon(x)), y)] \leq \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$$
$$+c\sqrt{\frac{2\ln(2e/\delta)}{n}} + \mathcal{B}_{g_\epsilon}(f) + \mathbb{1}\{f = \hat{f}^S\}\mathcal{G}_{\hat{f}^S},$$

where

$$\mathcal{B}_{g_\epsilon}(f) = \frac{1}{n}\sum_{y,k} |\mathcal{I}_k^y| \cdot \mathbb{E}_{x\sim\mathcal{D}_{x|y},\epsilon}\left[\ell(f(g_\epsilon(x)), y) - \ell(f(x), y) | x \in \mathcal{C}_k\right],$$

and $\mathcal{G}_{\hat{f}^S} = \sum_{y,k} \frac{2\mathcal{R}_{y,k}(\ell\circ\hat{\mathcal{F}})}{n} + c\sqrt{\frac{\ln(2e/\delta)}{2n}}$. Here, with independent uniform random variables $\xi_1, \ldots, \xi_n$ taking values in $\{-1, 1\}$, $\mathcal{R}_{y,k}(\ell \circ \hat{\mathcal{F}}) = \mathbb{E}_{\bar{S},\xi}[\sup_{\hat{f}\in\hat{\mathcal{F}}} \sum_{i=1}^{|\mathcal{I}_k^y|} \xi_i \ell(\hat{f}(\bar{x}_i), \bar{y}_i) | \bar{x}_i \in \mathcal{C}_k, \bar{y}_i = y]$. Moreover, if $x \in \mathcal{C}_k \Rightarrow g_\epsilon(x) \in \mathcal{C}_k$ with probability one, then $\mathcal{B}_{g_\epsilon}(f_\theta^S) = 0$.

Implications for Empirical Results: The first benefit of the use of key-value pairs appears in the training loss $\frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$ in the right-hand side (RHS) of the generalization error bound in Theorem 3.1. Since the bound is independent of the complexity of v_γ, we can set the hypothesis class of the local values v_γ to be highly expressive to reduce the training loss for each local region of the input space (in order to minimize the RHS of the bound in Theorem 3.1). Indeed, Figure 2 demonstrates this (although the main purpose is to show another benefit in terms of localization and forgetting). Moreover, the benefit of the proposed architecture is captured by two further mechanisms: The first mechanism, the term $\mathcal{B}_{g_\epsilon}(f)$ in Theorem 3.1 measures the effect of the input distribution shift g_ϵ. Theorem 3.1 shows that if the key of x is still the key of $g_\epsilon(x)$ (i.e., $x \in \mathcal{C}_k \Rightarrow g_\epsilon(x) \in \mathcal{C}_k$) with probability one, then we can further reduce the error as $\mathcal{B}_{g_\epsilon}(f_\theta^S) = 0$ while $\mathcal{B}_{g_\epsilon}(\hat{f}^S) > 0$ in general. This captures the mechanism, which is the minimization of the effect of the input distribution shifts via the discretization with key-value pairs. The second mechanism is shown by the fact that a model with the proposed architecture reduces the bound by making the last term $\mathbb{1}\{f = \hat{f}^S\}\mathcal{G}_{\hat{f}^S}$ vanish, because $\mathbb{1}\{f = \hat{f}^S\} = 0$ for $f = f_\theta^S$. This removed term consists of the complexity of its hypothesis class $\mathcal{R}_{y,k}(\ell \circ \hat{\mathcal{F}})$. Thus, the proposed model avoids the complexity cost as compared to other architectures. This is because the *bottleneck* with key-value pairs prevents over-fitting. Finally, the benefit from random projections in terms of Theorem 3.1 is that this projection can reduce the term of $\mathcal{B}_{g_\epsilon}(f)$ in addition to the flexibility in reducing the training loss term when having multiple codebooks as further discussed in Appendix B.

4. Related Work

The hallmark of machine learning is to be able to develop models that can quickly adapt to new tasks once trained on sufficiently diverse tasks (Baxter, 2000; Thrun & Pratt, 2012). Multiple ways to transfer information from one task to another exist: (1) transfer information via the transfer of the neural network weights (when trained on source tasks); (2) reuse raw data as in retrieval-based methods (Borgeaud et al., 2022; Nakano et al., 2021; Lee et al., 2019; Lewis et al., 2020; Guu et al., 2020; Sun et al., 2021; Goyal et al., 2022); or (3), via knowledge distillation (Hinton et al., 2015). Each approach implies inevitable trade-offs: When directly transferring neural network weights, previous information about the data may be lost in the fine-tuning process, while transfer via raw data may be prohibitively expensive as there can be hundreds of thousands of past experiences. At the same time, we want models that can continually accumulate information without training from scratch and forgetting previous information (Chen & Liu, 2018; Thrun, 1995; Davidson & Mozer, 2020). To prevent models from forgetting previous information, continual learning approaches make use of replay-buffers to replay old information (Rebuffi et al., 2017; Li & Hoiem, 2017; Shin et al., 2017; Tang et al., 2022), regularization during optimization (Kirkpatrick et al., 2017; Zenke et al., 2017), incrementally adding task-specific model components to increase capacity (Xiao et al., 2014), using parameter isolation-based approaches (Verma et al., 2021; Gao et al., 2022; Serra et al., 2018; Ke et al., 2021), or meta-learning fast-adapting models to promote quick adaptation (Harrison et al., 2020; He et al., 2020; Finn et al., 2019). The proposed method can be mostly linked with parameter isolation-based approaches by isolating the key-value pair for different tasks and classes. This bears some similarity with the theory of Sparse Distributed Memory (SDM), which is a model of associative memory inspired by human long-term memory and connections to neural circuits found in various organisms (Kanerva, 1988; 1992). The key-value bottleneck method has some similarities with the concurrent work by Bricken et al. (2023) which builds upon SDM ideas, but there are several crucial differences. While this model also writes and reads from a fixed set of "neurons", the proposed method does not require the implementation and careful tuning of additional neurobiology-inspired model components, such as "GABA switch implementations" to avoid "dead-neuron problems". Furthermore, key-value bottlenecks benefit from handling high-dimensional embedding spaces through random down-projection matrices and multiple distributed codebooks, while maintaining theoretical guarantees on catastrophic forgetting, thereby performing better on the presented task. The benefit of random down-projections is further supported by recent work exploring the effect of orthogonal projection-based methods as weight regularization to avoid forgetting (Zeng et al., 2019; Saha et al.,

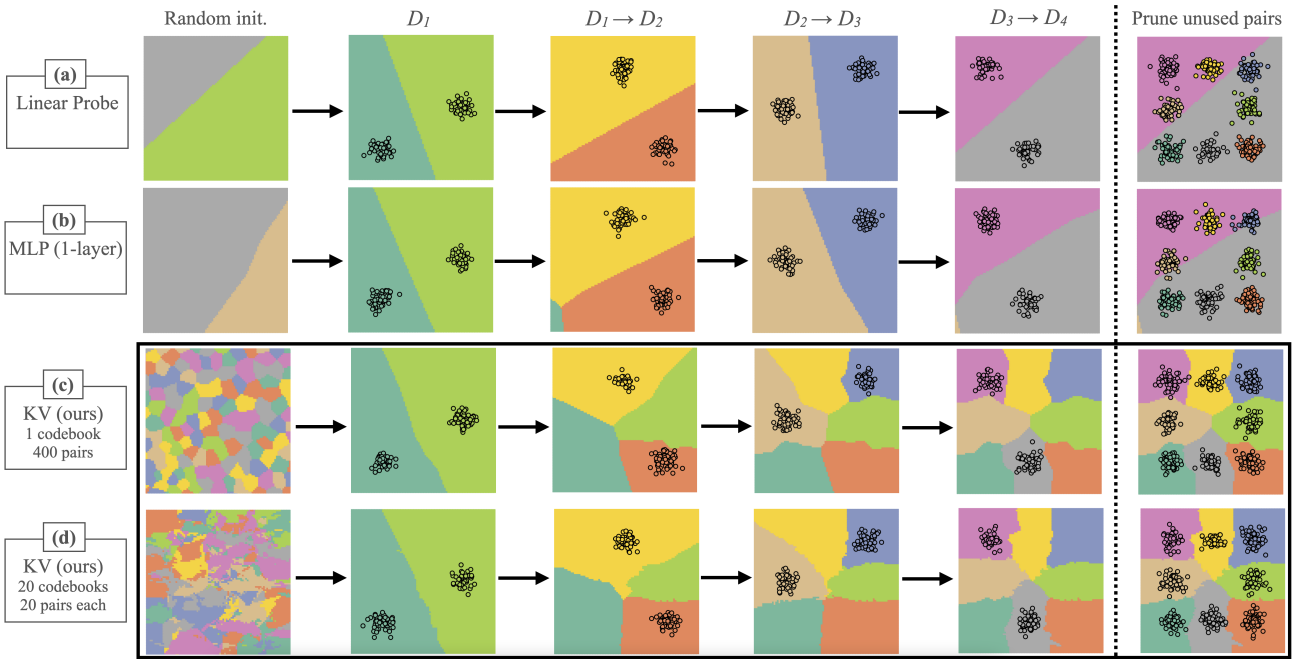

Figure 2. Incremental training on non-overlapping datasets $\mathcal{D}_1 \rightarrow \mathcal{D}_2 \rightarrow \mathcal{D}_3 \rightarrow \mathcal{D}_4$ (dots) using (a) Linear Probe (LP), (b) 1-layer MLP (MLP), (c) key-value bottleneck with C = 1 and ground truth feature heads, (d) key-value bottleneck with C = 20 and randomly projected feature heads. Using a discrete bottleneck of 400 key-value pairs with fixed key codes, and locally updated value codes, alleviates catastrophic forgetting. Color domains indicate one out of the 8 possible classes predicted by the model.

2021). Other models such as Shen et al. (2021) and Pourcel et al. (2022) focus on the online-learning setting without being exposed to the same classes for thousands of epochs. Most approaches for continual learning require the provision of task boundaries, but the proposed model architecture using discrete key-value codes avoids this constraint. The information is distilled into discrete key-value codes and a sparse set of value codes are locally adapted when faced with new tasks, which integrates information organically as discussed in Section 3 and has some parallels with local learning ideas (Bottou & Vapnik, 1992). A central part of the proposed method is the use of pre-trained models, a prerequisite that is also harnessed for continual learning tasks recently (Banayeeanzade et al., 2021; Ostapenko et al., 2022). The proposed method builds upon discrete representations, most notably the VQ-VAE (Van Den Oord et al., 2017; Liu et al., 2021) and various successor methods that have been shown to improve robustness under data corruptions (Razavi et al., 2019; Liu et al., 2021; Zeghidour et al., 2021; Yu et al., 2022; Shin et al., 2021; Mama et al., 2021). While these works can similarly reduce the representation space and enhance robustness, such bottlenecks have a single set of codes, which does not allow tuning them while keeping the feature-extracting backbone frozen. Finally, the key-value pairs can be interpreted as a memory bottleneck from which the model can selectively retrieve stored information. This interpretation bears similarity to memory networks, such as those proposed by (Sukhbaatar et al., 2015; Chandar et al., 2016; Webb et al., 2021; Lample et al., 2019; Panigrahy et al., 2021), which also write and read information from a set of symbol-like memory cells. However, key-value codes act as an information bottleneck, whereas memory networks condition predictions on retrieved memory.

5. Experiments

The insight we aim to convey with the presented experiments is to validate that having a key-value bottleneck allows for *localized, context-dependent model updates* that are robust to training under input distribution changes. We will show that this allows us to *reduce catastrophic forgetting*; *integrate new information*; and thereby *mitigate vulnerabilities* in challenging class-incremental learning settings. We first showcase a simplified yet challenging class-incremental learning setting to give readers an understanding of why standard approaches fail and how the proposed method succeeds. We then explore the method's benefits under a challenging class-incremental learning scenario on CIFAR10 and analyze the role of different components and design choices of a key-value bottleneck. The code for reproducing all experiments can be found in the supplementary material.

5.1. A Simple Learning Setting Motivating the Method

We consider a 2D input feature classification problem for 8 classes, where the training data is not i.i.d. but changes over four stages (see Figure 2). In each stage, we sample 100 examples of two classes for 1000 training steps, using gradient descent to update the weights, then move on to two new classes for the next 1000 steps. The input features of each class follow spatially separated normal distributions. We first discuss the behaviour on this task using a vanilla linear probe or 1-layer MLP with 32 hidden dimensions trained by minimizing the cross-entropy along the training data stream. As can be seen from the decision boundaries in Figure 2 (a, b), these naive approaches — unsurprisingly — over-fit on the most recent training data. This behaviour is because the MLP is optimizing *all* weights on the most recent training data, thereby overwriting prior predictions. This problem can be easily alleviated by *sparse, local* updates using a discrete key-value bottleneck. Here we present two different configurations. In the first configuration, we use 400 pairs of 2-dim keys and 8-dim values without random projection and we initialize these keys from random points in the domain $x \in [0, 1]^2$ (not the target dataset). Input x_i will now snap to the closest key and fetch its corresponding value. After keys are initialized, they are frozen, and we train the model on the non-stationary data stream using the same loss and optimizer as before but optimizing the values only. As can be seen from Figure 2 (c), the proposed bottleneck mechanism will now solely update the input-dependent value codes. This enables the model to incrementally update its predictions on unseen input domains and minimize interference. Finally, we can prune all key-value pairs that were never selected in order to remove any dormant key-value codes (rightmost column). In the above example, it is essential to have two separate sets of codes compared to a single set of discrete codes. Finally, we also study a configuration with 20 codebooks each with 20 key-value pairs and use randomly initialized projection matrices. This configuration, which uses multiple codebooks but the same total number of key-value pairs, works just as well as seen in Figure 2 (d).

5.2. Continual Learning on Real-World Data

Having established some first intuition on the localized model updates, we now attempt to present the strong learning capabilities of the introduced bottleneck on a challenging and realistic continual class-incremental learning task.

Experimental Setup. We use a class-incremental CIFAR10 task with pre-training on another dataset without any memory replay or provision of task boundaries, which is a very challenging task setting in continual learning. Here, five disjoint sets with two classes each are incrementally presented for many epochs each and the goal is to learn new classes and not forget previous classes at the very end of the training. We present each set for 2000 epochs, which in an ordinary architecture would cause catastrophic forgetting of the previous sets. We perform five replications of each model with different random seeds, with the seed also re-sampling the class splits to avoid selecting any favourable class split. Importantly, we study one of the hardest yet most realistic settings in continual learning by *not allowing any memory replay or provision of task boundaries* which is required by the vast majority of existing continual learning methods. To the best of our knowledge, the only method that can deal with this learning scenario is (Bricken et al., 2023). We report experiments on five publicly available backbones including:

1. ResNet50 pre-trained on ImageNet
2. ViT-B/32 pre-trained with CLIP
3. ResNet50w2 pre-trained with SwAV as SSL method
4. ResNet50 pre-trained with DINO as SSL method
5. ConvMixer pre-trained on 32x32 downsampled Imagenet. We select this backbone to directly compare against results reported in (Bricken et al., 2023).

We initialize keys on the unlabelled non-overlapping CIFAR100 dataset except for the ConvMixer where we used the embeddings from the downsampled Imagenet dataset for reasons of comparison. We test the wide applicability of key-value bottlenecks across these different architectures and pre-training schemes, using 256 codebooks with 4096 key-value pairs each, 14-dimensional keys and 10-dimensional values. We additionally report results for the proposed bottleneck and a Linear Probe under an IID training scheme (presenting all classes simultaneously), with the latter representing a fair oracle upper bound. When learning class-incrementally, we compare against the reported SDMLP performance on the ConvMixer backbone, as well as a 1-layer MLP with 128 hidden dimensions and a linear probe with and without bias term for all backbones. We refer to Appendix D for further experiments.

Results. We summarize our results in Figure 3. First, we observe that the key-value bottleneck can indeed successfully learn and integrate new information under such a difficult class-incremental distribution shift consistently (green curves). At the same time, the key-value bottleneck substantially reduces the severe catastrophic forgetting over the standard adaptation approaches (red, brown and purple curves). In line with the theory, we observe *across all encoder backbones* the appealing learning property that the bottleneck yields the same performance at the end of training compared to the benign i.i.d. setting (blue lines). Within the class incremental setting only, we observe that the *final performance* is almost identical, irrespective of the sampled class split. This result suggests that the bottleneck can

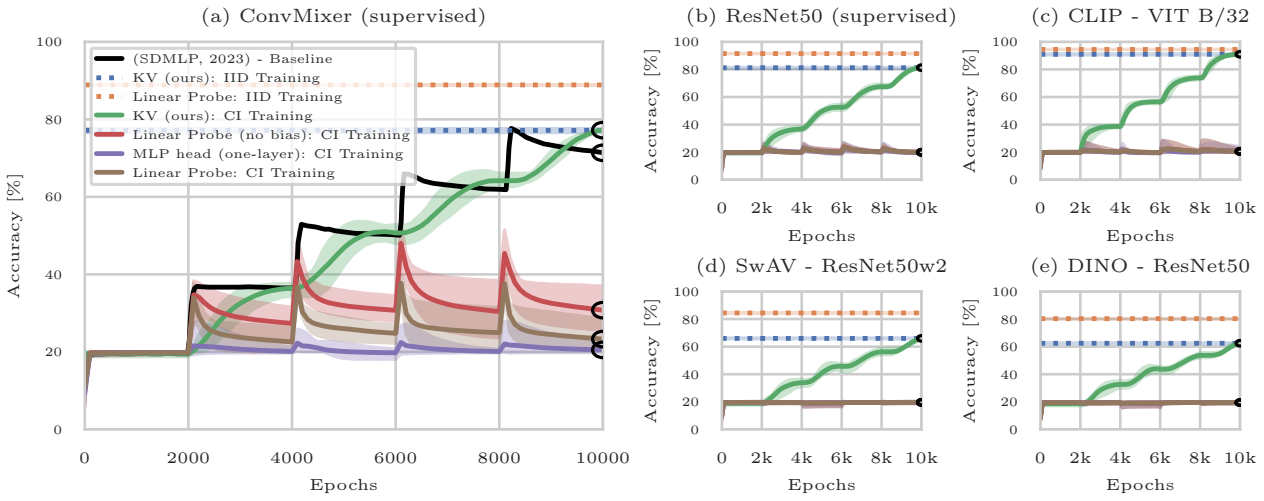

Figure 3. The key-value bottleneck architecture successfully learns and integrates new information in a challenging 5-split CIFAR10 class-incremental (CI) learning setting. Our method (green line) reduces catastrophic forgetting and yields strong final performance across various pre-trained frozen backbone architectures including (a) ConvMixer; (b) ResNet50 ; (c) ViT B/32 with CLIP, (d) ResNet50w2 with SwAV; (e) ResNet50 with DINO. At the end of the training, the proposed method outperforms the reported performance of the SDM method using the same pre-trained ConvMixer. Standard probe tuning exhibits major forgetting in this scenario (red, brown, purple curve).

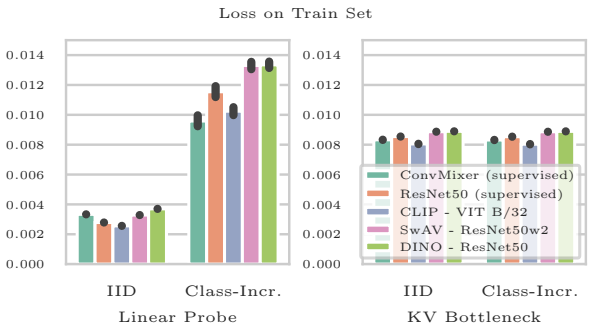

Figure 4. We demonstrate that the proposed key-value bottleneck method maintains a constant loss measure, directly translating to improved generalization when training class incrementally, as opposed to standard approaches which can over-fit.

successfully mitigate the typically very destructive effect of learning under non-stationary training data and might be particularly robust against training curricula that are otherwise adversarial to stochastic gradient-based training. We can partially explain this behaviour by the fact that we freeze the keys during training and use a simple momentum-free SGD optimizer. In this case, any value code will approximatively receive the same gradients over the entire course of training.

We now take a look at the results of the ConvMixer backbone in Figure 3 (a). This reflects the same setup as in the SDMLP method, the only applicable method we are aware of in this challenging task setting without memory replay or provision of task boundaries. The key-value bottleneck method achieves a final accuracy of 77.3% compared to 71% for the best final accuracy with SDMLP. Additionally, the learning curves with the bottleneck show smooth improvements in performance, whereas the baseline spikes at the introduction of new tasks and exhibits pronounced catastrophic forgetting. Furthermore, we observe consistent results across all other four backbones. The best results are achieved with a CLIP-encoder with 91.3% and the supervised pre-trained ResNet50 backbone with 81.5% final performance (Figure 3 b & c). We are similarly getting strong class-incremental learning abilities for both SSL backbones, which were pre-trained without any labels, achieving up to 66.3% in the case of SwAV and 62.5% in the case of DINO (Figure 3 d & e). Finally, we believe these results are further encouraging as they demonstrate that the key-value bottleneck is able to achieve strong performance across a wide variety of pre-trained backbones and learned feature spaces, suggesting that it has the potential to scale to even more performant models in the future. Finally, we show in Figure 4 that a key-value bottleneck does not over-fit like other standard approaches. A linear probe may perform better when trained on all classes at once in an i.i.d. setting; but when trained incrementally, it over-fits. However, the key-value bottleneck maintains a constant loss measure over both, i.i.d. and class-incremental settings, which leads to better overall performance when training on new classes. This aligns with our analysis of the first term in Theorem 3.1.

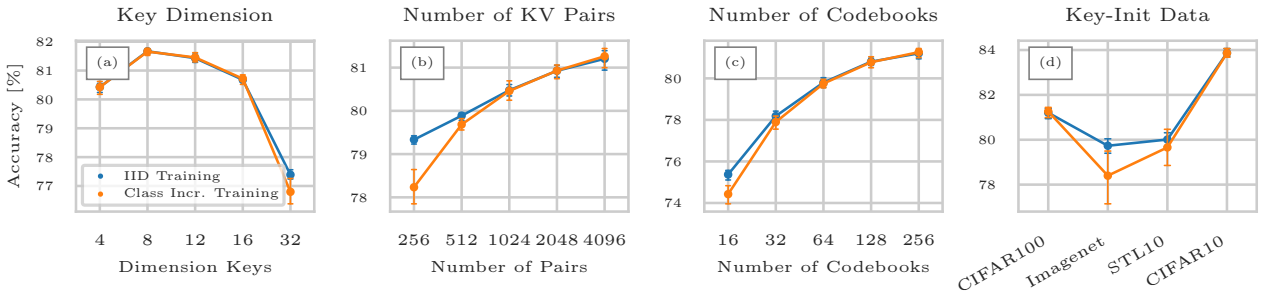

Figure 5. Assessing the role of individual model components starting from the base model architecture in Figure 3 b. Reducing the dimensionality of keys has a positive effect (a); increasing the number of pairs and codebooks enhances generalization performance (b + c); finally, key-value bottlenecks keys can be initialized from a different dataset without major drop in overall performance (d).

5.3. Ablation and Sensitivity Analysis

We now analyze the role of various relevant bottleneck components to better understand its learning abilities, as well as further opportunities and limitations. We point the reader to Appendix E.3 for an in-depth analysis of the (key,value) codes learned by the models. In the following paragraphs, we focus on the ResNet50 backbone and analyze the effect of changing one particular component while keeping everything else fixed. We present the results in Figure 5.

Dimension of Key Codes. First, we found that there is an optimal number of dimensions for the keys, around 8 to 12, which can be explained as follows. Having keys with high dimensions requires more data to cover the backbone manifold sufficiently. On the other hand, having keys with too few dimensions means that the keys are scattered in a much smaller space which increases the chances of unintended key sharing among different classes.

Number of Key-Value Pairs. The number of keys used in the bottleneck can greatly affect the performance of the model. We found that increasing the number of key-value pairs leads to better performance, with only a small decrease in performance when using four times fewer pairs. Our analysis also suggests that the model performance will continue to improve asymptotically, though we have not yet reached that point. When using 256 or fewer key-value pairs, we noticed a separation between the performance of the model in i.i.d. and class-incremental settings. This is likely because when there are fewer pairs, more of them are shared among multiple classes, and this allows for a more balanced update of the gradients when training on mini-batches.

Number of Codebooks. The number of codebooks is another important factor that controls the capacity of the model, similar to the number of key-value pairs. We observed that using more codebooks leads to better performance, with a slightly more pronounced effect compared to the number of pairs. Furthermore, increasing the number of codebooks also decreases the training loss term. However, it is important to note that increasing the number of codebooks too much can also increase the value of c in Theorem 3.1, although we have not yet observed that in our experiments.

Key Initialization Dataset. Finally, we tested the robustness of the proposed method by initializing the keys on different datasets, such as STL10 and Imagenet, in addition to CIFAR100 used in previous experiments (Coates et al., 2011; Krizhevsky et al., 2009). We also calculated the performance of an "oracle" key initialization by using the unlabelled CIFAR10 dataset. While there is a slight decrease in performance compared to the oracle, the architecture is robust to different datasets, even if those datasets differ in resolution and scenes.

6. Limitations and Extensions

Pre-trained Encoders. The key-value bottleneck method relies on pre-trained encoders that can extract meaningful features shared between the pre-training and actual training data. This is an important requirement that has been shown by prior works in transfer learning (Yosinski et al., 2014; Chen et al., 2020; Azizi et al., 2021). However, as self-supervised pre-trained models from massive datasets become more prevalent, a key-value bottleneck can be used to connect these encoders to downstream tasks and minimize forgetting under distribution changes. Additionally, the method relies on encoders that are able to separate distinct features locally, as supported by recent research (Huang et al., 2019; Caron et al., 2021; Zhou et al., 2022a; Tian et al., 2022). Fine-tuning the encoder directly can distort these features and, therefore, negatively affect out-of-distribution generalization (Kumar et al., 2022). The key-value bottleneck reduces this issue as encoder features are not influenced by fine-tuning.

Selection and Initialization of Key Codes. The selection and initialization of key codes is an important aspect of the proposed method. The approach of using EMA works well for a range of datasets, but it may have limitations in extreme situations where the dataset is unknown or changes completely, such as in RL. We did not explore these scenarios in this work, but we believe that discrete key-value bottlenecks have the potential to be extended to handle them. For example, the encoder, projection, and keys can be fine-tuned asynchronously with auxiliary unsupervised tasks to handle distribution shifts. Additionally, information about the distance between the codebook input heads and their closest keys can be used to estimate the level of distribution shift and adapt, add, or reinitialize key-value pairs as needed.

Decoder Architecture When classifying, we found it effective to use a non-parametric decoder without learnable weights. Nevertheless, we can also train a discrete key-value bottleneck with more complex, parametric decoders. When training the model under i.i.d. conditions, we can adjust the decoder weights to improve performance. However, when the training conditions change, e.g. in incremental learning, we need to be more careful. One could make the decoder weights adjustable and augment with various regularization methods or keep them fixed if necessary. Combining key-value bottlenecks with advanced, pre-trained decoders such as generative models is a promising area for future research.

Trade-offs using a Bottleneck. Using an information bottleneck involves a trade-off between accuracy and information compression. The proposed bottleneck applies strong compression by propagating only a discrete set of C key-value pairs, preventing changes to the encoder. This allows for training under strong distribution shifts without forgetting, but with slightly lower performance. If the target data distribution is available before (no labels needed), we can improve performance by initializing the keys on this distribution, but there are also other ways to potentially improve further such as parametric decoders or allowing back-propagation into the projection weights and keys through techniques such as sampling or straight-through estimators.

7. Conclusion

We proposed a discrete key-value bottleneck architecture that can adapt to input distribution shifts and performs well in class-incremental learning settings. We supported the model's effectiveness theoretically and through empirical validation. Specifically, we showed that the use of key-value pairs in the architecture can reduce the generalization error and prevent over-fitting. The model is able to integrate new information locally while avoiding catastrophic forgetting. We believe the key-value bottleneck could be a promising solution for tackling challenging training scenarios.

Acknowledgements

We would like to thank Jonas Wildberger, Andrea Banino, Andrea Dittadi, Diego Agudelo España, Chiyuan Zhang, Arthur Szlam, Olivier Tieleman, Wisdom D'Almeida, Felix Leeb, Alex Lamb, Xu Ji, Dianbo Liu, Aniket Didolar, Nan Rosemary Ke and Moksh Jain for valuable feedback. The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting FT.

References

Azizi, S., Mustafa, B., Ryan, F., Beaver, Z., Freyberg, J., Deaton, J., Loh, A., Karthikesalingam, A., Kornblith, S., Chen, T., et al. Big self-supervised models advance medical image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3478–3488, 2021.

Banayeeanzade, M., Mirzaiezadeh, R., Hasani, H., and Soleymani, M. Generative vs. discriminative: Rethinking the meta-continual learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Baxter, J. A model of inductive bias learning. *Journal of artificial intelligence research*, 12:149–198, 2000.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., Van Den Driessche, G. B., Lespiau, J.-B., Damoc, B., Clark, A., et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pp. 2206–2240. PMLR, 2022.

Bottou, L. and Vapnik, V. Local learning algorithms. *Neural computation*, 4(6):888–900, 1992.

Bricken, T., Davies, X., Singh, D., Krotov, D., and Kreiman, G. Sparse distributed memory is a continual learner. *International Conference for Learning Representations (ICLR)*, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., and Joulin, A. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in Neural Information Processing Systems*, 33:9912–9924, 2020.

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. Emerging properties in self-supervised vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

Chandar, S., Ahn, S., Larochelle, H., Vincent, P., Tesauro, G., and Bengio, Y. Hierarchical memory networks. *arXiv preprint arXiv:1605.07427*, 2016.

Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. E. Big self-supervised models are strong semi-supervised learners. *Advances in neural information processing systems*, 33:22243–22255, 2020.

Chen, Z. and Liu, B. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018.

Coates, A., Ng, A., and Lee, H. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223. JMLR Workshop and Conference Proceedings, 2011.

Davidson, G. and Mozer, M. C. Sequential mastery of multiple visual tasks: Networks naturally learn to learn and forget to forget. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9282–9293, 2020.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Finn, C., Rajeswaran, A., Kakade, S., and Levine, S. Online meta-learning. In *International Conference on Machine Learning*, pp. 1920–1930. PMLR, 2019.

Gao, Q., Luo, Z., Klabjan, D., and Zhang, F. Efficient architecture search for continual learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Goyal, A., Friesen, A., Banino, A., Weber, T., Ke, N. R., Badia, A. P., Guez, A., Mirza, M., Humphreys, P. C., Konyushova, K., et al. Retrieval-augmented reinforcement learning. In *International Conference on Machine Learning*, pp. 7740–7765. PMLR, 2022.

Guu, K., Lee, K., Tung, Z., Pasupat, P., and Chang, M. Retrieval augmented language model pre-training. In *International conference on machine learning*, pp. 3929–3938. PMLR, 2020.

Harrison, J., Sharma, A., Finn, C., and Pavone, M. Continuous meta-learning without tasks. *Advances in neural information processing systems*, 33:17571–17581, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, X., Sygnowski, J., Galashov, A., Rusu, A. A., Teh, Y. W., and Pascanu, R. Task agnostic continual learning via meta learning. In *4th Lifelong Machine Learning Workshop at ICML 2020*, 2020.

Hinton, G., Vinyals, O., Dean, J., et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2019.

Huang, J., Dong, Q., Gong, S., and Zhu, X. Unsupervised deep learning by neighbourhood discovery. In *International Conference on Machine Learning*, pp. 2849–2858. PMLR, 2019.

Kanerva, P. *Sparse distributed memory*. MIT press, 1988.

Kanerva, P. Sparse distributed memory and related models. Technical report, 1992.

Kawaguchi, K., Deng, Z., Luh, K., and Huang, J. Robustness implies generalization via data-dependent generalization bounds. In *International Conference on Machine Learning*, pp. 10866–10894. PMLR, 2022.

Ke, Z., Liu, B., Ma, N., Xu, H., and Shu, L. Achieving forgetting prevention and knowledge transfer in continual learning. *Advances in Neural Information Processing Systems*, 34:22443–22456, 2021.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Kumar, A., Raghunathan, A., Jones, R., Ma, T., and Liang, P. Fine-tuning can distort pretrained features and underperform out-of-distribution. *International Conference on Learning Representations*, 2022.

Lample, G., Sablayrolles, A., Ranzato, M., Denoyer, L., and Jégou, H. Large memory layers with product keys. *Advances in Neural Information Processing Systems*, 32, 2019.

Lee, K., Chang, M.-W., and Toutanova, K. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300*, 2019.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

Li, Z. and Hoiem, D. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Liu, D., Lamb, A. M., Kawaguchi, K., Goyal, A., Sun, C., Mozer, M. C., and Bengio, Y. Discrete-valued neural communication. *Advances in Neural Information Processing Systems*, 34, 2021.

Mama, R., Tyndel, M. S., Kadhim, H., Clifford, C., and Thurairatnam, R. Nwt: Towards natural audio-to-video generation with representation learning. *arXiv preprint arXiv:2106.04283*, 2021.

Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Ostapenko, O., Lesort, T., Rodríguez, P., Arefin, M. R., Douillard, A., Rish, I., and Charlin, L. Foundational models for continual learning: An empirical study of latent replay. *arXiv preprint arXiv:2205.00329*, 2022.

Panigrahy, R., Wang, X., and Zaheer, M. Sketch based memory for neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 3169–3177. PMLR, 2021.

Pham, H., Dai, Z., Ghiasi, G., Kawaguchi, K., Liu, H., Yu, A. W., Yu, J., Chen, Y.-T., Luong, M.-T., Wu, Y., et al. Combined scaling for open-vocabulary image classification. *arXiv e-prints*, pp. arXiv–2111, 2021.

Pourcel, J., Vu, N.-S., and French, R. M. Online task-free continual learning with dynamic sparse distributed memory. In *European Conference on Computer Vision*, pp. 739–756. Springer, 2022.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pp. 8748–8763. PMLR, 2021.

Razavi, A., Van den Oord, A., and Vinyals, O. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3): 211–252, 2015.

Saha, G., Garg, I., and Roy, K. Gradient projection memory for continual learning. *International Conference for Learning Representations (ICLR)*, 2021.

Serra, J., Suris, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pp. 4548–4557. PMLR, 2018.

Shen, Y., Dasgupta, S., and Navlakha, S. Algorithmic insights on continual learning from fruit flies. *arXiv preprint arXiv:2107.07617*, 2021.

Shin, H., Lee, J. K., Kim, J., and Kim, J. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017.

Shin, W., Lee, G., Lee, J., Lee, J., and Choi, E. Translation-equivariant image quantizer for bi-directional image-text generation. *arXiv preprint arXiv:2112.00384*, 2021.

Sukhbaatar, S., Weston, J., Fergus, R., et al. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015.

Sun, Y., Wang, S., Feng, S., Ding, S., Pang, C., Shang, J., Liu, J., Chen, X., Zhao, Y., Lu, Y., et al. Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation. *arXiv preprint arXiv:2107.02137*, 2021.

Tang, Y.-M., Peng, Y.-X., and Zheng, W.-S. Learning to imagine: Diversify memory for incremental learning using unlabeled data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9549–9558, 2022.

Thrun, S. Is learning the n-th thing any easier than learning the first? *Advances in neural information processing systems*, 8, 1995.

Thrun, S. and Pratt, L. *Learning to learn*. Springer Science & Business Media, 2012.

Tian, R., Wu, Z., Dai, Q., Hu, H., and Jiang, Y. Deeper insights into vits robustness towards common corruptions. *arXiv preprint arXiv:2204.12143*, 2022.

Trockman, A. and Kolter, J. Z. Patches are all you need? *arXiv preprint arXiv:2201.09792*, 2022.

Van de Ven, G. M. and Tolias, A. S. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

Van Den Oord, A., Vinyals, O., et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Verma, V. K., Liang, K. J., Mehta, N., Rai, P., and Carin, L. Efficient feature transformations for discriminative and generative continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13865–13875, 2021.

Wang, Z., Zhang, Z., Lee, C.-Y., Zhang, H., Sun, R., Ren, X., Su, G., Perot, V., Dy, J., and Pfister, T. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 139–149, 2022.

Webb, T. W., Sinha, I., and Cohen, J. D. Emergent symbols through binding in external memory. *International Conference on Learning Representations*, 2021.

Xiao, T., Zhang, J., Yang, K., Peng, Y., and Zhang, Z. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 177–186, 2014.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

Yu, J., Li, X., Koh, J. Y., Zhang, H., Pang, R., Qin, J., Ku, A., Xu, Y., Baldridge, J., and Wu, Y. Vector-quantized image modeling with improved vqgan. *International Conference on Learning Representations*, 2022.

Zeghidour, N., Luebs, A., Omran, A., Skoglund, J., and Tagliasacchi, M. Soundstream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2021.

Zeng, G., Chen, Y., Cui, B., and Yu, S. Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 1(8):364–372, 2019.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pp. 3987–3995. PMLR, 2017.

Zhou, D., Yu, Z., Xie, E., Xiao, C., Anandkumar, A., Feng, J., and Alvarez, J. M. Understanding the robustness in vision transformers. In *International Conference on Machine Learning*, pp. 27378–27394. PMLR, 2022a.

Zhou, K., Yang, J., Loy, C. C., and Liu, Z. Learning to prompt for vision-language models. *International Journal of Computer Vision*, 130(9):2337–2348, 2022b.

A. Proof of Theorem 3.1

Proof. Let $f \in \{f_\theta^S, \hat{f}^S\}$. We define

$$\mathcal{C}_k^y = \left\{ (\bar{x}, \bar{y}) \in \mathcal{X} \times \mathcal{Y} : y = \bar{y}, k = \operatorname*{argmin}_{i \in [|\mathcal{K}|]} d((k_\beta \circ z_\alpha)(\bar{x}), \mathcal{K}_i) \right\}.$$

We use the notation of $z = (x, y)$ and $\tilde{x}^\epsilon = g_\epsilon(x)$. We first write the expected error as the sum of the conditional expected error:

$$\mathbb{E}_z[\ell(f(\tilde{x}^\epsilon), y)] = \sum_{k,y} \mathbb{E}_x[\ell(f(\tilde{x}^\epsilon), y) | z \in \mathcal{C}_k^y] \Pr(z \in \mathcal{C}_k^y) = \sum_{k,y} \mathbb{E}_{x_{k,y}}[\ell(f(\tilde{x}_{k,y}^\epsilon), y)] \Pr(z \in \mathcal{C}_k^y),$$

where $\tilde{x}_{k,y}^\epsilon = g_\epsilon(x_{k,y})$ and $x_{k,y}$ is the random variable x conditioned on $z \in \mathcal{C}_k^y$. Using this, we decompose the generalization error into two terms:

$$\begin{aligned} \mathbb{E}_z[\ell(f(\tilde{x}^\epsilon), y)] - \frac{1}{n}\sum_{i=1}^n \ell(f(x_i), y_i) &= \sum_{k,y} \mathbb{E}_{x_{k,y}}[\ell(f(\tilde{x}_{k,y}^\epsilon), y)] \left(\Pr(z \in \mathcal{C}_k^y) - \frac{|\mathcal{I}_k^y|}{n} \right) \\ &+ \left(\sum_{k,y} \mathbb{E}_{x_{k,y}}[\ell(f(\tilde{x}_{k,y}^\epsilon), y)] \frac{|\mathcal{I}_k^y|}{n} - \frac{1}{n}\sum_{i=1}^n \ell(f(x_i), y_i) \right). \end{aligned} \tag{2}$$

Since $\frac{1}{n}\sum_{i=1}^n \ell(h, r_i) = \frac{1}{n}\sum_y \sum_{k \in I_\mathcal{K}^y} \sum_{i \in \mathcal{I}_k^y} \ell(f(x_i), y_i)$, the second term in the right-hand side of (2) is further simplified as

$$\begin{aligned} &\sum_{k,y} \mathbb{E}_{x_{k,y}}[\ell(f(\tilde{x}_{k,y}^\epsilon), y)] \frac{|\mathcal{I}_k^y|}{n} - \frac{1}{n}\sum_{i=1}^n \ell(f(x_i), y_i) \\ &= \frac{1}{n}\sum_y \sum_{k \in I_\mathcal{K}^y} |\mathcal{I}_k^y| \left(\mathbb{E}_{x_{k,y}}[\ell(f(\tilde{x}_{k,y}^\epsilon), y)] - \frac{1}{|\mathcal{I}_k^y|} \sum_{i \in \mathcal{I}_k^y} \ell(f(x_i), y_i) \right). \end{aligned}$$

where $I_\mathcal{K}^y = \{k \in [|\mathcal{K}|] : |\mathcal{I}_k^y| \geq 1\}$. Substituting these into equation (2) yields

$$\begin{aligned} \mathbb{E}_z[\ell(f(\tilde{x}^\epsilon), y)] - \frac{1}{n}\sum_{i=1}^n \ell(f(x_i), y_i) &= \sum_{k,y} \mathbb{E}_{x_{k,y}}[\ell(f(\tilde{x}_k^\epsilon), y)] \left(\Pr(z \in \mathcal{C}_k^y) - \frac{|\mathcal{I}_k^y|}{n} \right) \\ &+ \frac{1}{n}\sum_y \sum_{k \in I_\mathcal{K}^y} |\mathcal{I}_k^y| \left(\mathbb{E}_{x_{k,y}}[\ell(f(\tilde{x}_{k,y}^\epsilon), y)] - \frac{1}{|\mathcal{I}_k^y|} \sum_{i \in \mathcal{I}_k^y} \ell(f(x_i), y_i) \right) \end{aligned} \tag{3}$$

By using Lemma 1 of (Kawaguchi et al., 2022), we have that for any $\delta > 0$, with probability at least $1 - \delta$,

$$\begin{aligned} &\sum_{k,y} \mathbb{E}_{x_{k,y}}[\ell(f(\tilde{x}_k^\epsilon), y)] \left(\Pr(z \in \mathcal{C}_k^y) - \frac{|\mathcal{I}_k^y|}{n} \right) \\ &\leq \left(\sum_{k,y} \mathbb{E}_{x_{k,y}}[\ell(f(\tilde{x}_k^\epsilon), y)] \sqrt{\Pr(z \in \mathcal{C}_k^y)} \right) \sqrt{\frac{2\ln(|\mathcal{Y}||\mathcal{K}|/\delta)}{n}} \\ &\leq M \left(\sum_{k,y} \sqrt{\Pr(z \in \mathcal{C}_k^y)} \right) \sqrt{\frac{2\ln(|\mathcal{Y}||\mathcal{K}|/\delta)}{n}}. \end{aligned} \tag{4}$$

Here, we have that $\ln(|\mathcal{Y}||\mathcal{K}|/\delta) = \ln(e^{\ln|\mathcal{Y}||\mathcal{K}|}/\delta) \leq \ln(e^\zeta/\delta) \leq \ln((e/\delta)^\zeta) = \zeta\ln(e/\delta)$ where $\zeta = \max(1, \ln|\mathcal{Y}||\mathcal{K}|)$, since $\delta \in (0, 1)$ and $\zeta \geq 1$. Moreover, note that for any (f, h, M) such that $M > 0$ and $B \geq 0$ for all X, we have that $\mathbb{P}(f(X) \geq M) \geq \mathbb{P}(f(X) > M) \geq \mathbb{P}(Bf(X) + h(X) > BM + h(X))$, where the probability is with respect to the

randomness of X. Thus, by combining (3) and (4), we have that for any $\delta > 0$, with probability at least $1 - \delta$, the following holds:

$$
\begin{aligned}
&\mathbb{E}_{z,\epsilon}[\ell(f(g_\epsilon(x)), y)] - \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i) \qquad (5)\\
&= \mathbb{E}_\epsilon \left[\mathbb{E}_z[\ell(f(g_\epsilon(x)), y)] - \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i) \right] \\
&\leq \frac{1}{n}\sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y} |\mathcal{I}_k^y| \left(\mathbb{E}_{z,\epsilon}[\ell(f(g_\epsilon(x)), y)|z \in \mathcal{C}_k^y] - \frac{1}{|\mathcal{I}_k^y|}\sum_{i\in\mathcal{I}_k^y} \ell(f(x_i), y_i) \right) + c\sqrt{\frac{2\ln(e/\delta)}{n}} \\
&= \frac{1}{n}\sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y} |\mathcal{I}_k^y| \left(\mathbb{E}_{z}[\ell(f(x), y)|z \in \mathcal{C}_k^y] - \frac{1}{|\mathcal{I}_k^y|}\sum_{i\in\mathcal{I}_k^y} \ell(f(x_i), y_i) \right) \\
&\quad + \frac{1}{n}\sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y} |\mathcal{I}_k^y| \mathbb{E}_z \left[\mathbb{E}_\epsilon[\ell(f(g_\epsilon(x)), y)] - \ell(f(x), y)|z \in \mathcal{C}_k^y \right] \\
&\quad + c\sqrt{\frac{2\ln(e/\delta)}{n}}
\end{aligned}
$$

We now bound each term in the right-hand side of equation (5) for both cases of $f = \hat{f}^S$ and $f = f_\theta^S$. We first consider the case of $f = \hat{f}^S$. For the first term in the right-hand side of equation (5), we invoke Lemma 4 of (Pham et al., 2021) to obtain that for any $\delta > 0$, with probability at least $1 - \delta$,

$$
\begin{aligned}
&\frac{1}{n}\sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y} |\mathcal{I}_k^y| \left(\mathbb{E}_{z}[\ell(\hat{f}^S(x), y)|z \in \mathcal{C}_k^y] - \frac{1}{|\mathcal{I}_k^y|}\sum_{i\in\mathcal{I}_k^y} \ell(\hat{f}^S(x_i), y_i) \right) \\
&\leq \frac{1}{n}\sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y} |\mathcal{I}_k^y| \left(2\mathcal{R}_{y,k}(\ell\circ\hat{\mathcal{F}}) + M\sqrt{\frac{\ln(|\mathcal{Y}||\mathcal{K}|/\delta)}{2|\mathcal{I}_k^y|}} \right) \\
&= 2\sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y} |\mathcal{I}_k^y| \frac{\mathcal{R}_{y,k}(\ell\circ\hat{\mathcal{F}})}{n} + M\sqrt{\frac{\ln(|\mathcal{Y}||\mathcal{K}|/\delta)}{2n}} \sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y} \sqrt{\frac{|\mathcal{I}_k^y|}{n}}
\end{aligned}
$$

where we used the fact that $\sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y} |\mathcal{I}_k^y| = n$. Moreover, we have that using the Cauchy–Schwarz inequality,

$$
\sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y} \sqrt{\frac{|\mathcal{I}_k^y|}{n}} \leq \sqrt{\sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y} \frac{|\mathcal{I}_k^y|}{n}} \sqrt{\sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y} 1} \leq \sqrt{|\mathcal{Y}||\mathcal{K}|}.
$$

Thus, since $\ln(|\mathcal{Y}||\mathcal{K}|/\delta) \leq \zeta \ln(e/\delta)$ with $\zeta = \max(1, \ln|\mathcal{Y}||\mathcal{K}|)$ (see above), for any $\delta > 0$, with probability at least $1 - \delta$,

$$
\begin{aligned}
&\frac{1}{n}\sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y} |\mathcal{I}_k^y| \left(\mathbb{E}_{z}[\ell(\hat{f}^S(x), y)|z \in \mathcal{C}_k^y] - \frac{1}{|\mathcal{I}_k^y|}\sum_{i\in\mathcal{I}_k^y} \ell(\hat{f}^S(x_i), y_i) \right) \qquad (6)\\
&\leq 2\sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y} |\mathcal{I}_k^y| \frac{\mathcal{R}_{y,k}(\ell\circ\hat{\mathcal{F}})}{n} + c\sqrt{\frac{\ln(e/\delta)}{2n}}.
\end{aligned}
$$

For the case of $f = \hat{f}^S$, by combining equation (5) and (6) with union bound, it holds that any $\delta > 0$, with probability at

least $1-\delta$,

$$\begin{aligned}
&\mathbb{E}_{z,\epsilon}[\ell(\hat{f}^S(g_\epsilon(x)),y)] - \frac{1}{n}\sum_{i=1}^{n}\ell(\hat{f}^S(x_i),y_i) \\
&\leq 2\sum_{y\in\mathcal{Y}}\sum_{k\in I_{\mathcal{K}}^y}|\mathcal{I}_k^y|\frac{\mathcal{R}_{y,k}(\ell\circ\hat{\mathcal{F}})}{n} + c\sqrt{\frac{\ln(2e/\delta)}{2n}} + c\sqrt{\frac{2\ln(2e/\delta)}{n}} \\
&\quad + \frac{1}{n}\sum_{y,k}|\mathcal{I}_k^y|\mathbb{E}_{x\sim\mathcal{D}_{x|y},\epsilon}\left[\ell(\hat{f}^S(g_\epsilon(x)),y) - \ell(\hat{f}^S(x),y)|x\in\mathcal{C}_k\right].
\end{aligned}$$

We now consider the case of $f = f_\theta^S$. For the first term in the right-hand side of equation (5),

$$\begin{aligned}
&\mathbb{E}_z[\ell(f_\theta^S(x),y)|z\in\mathcal{C}_k^y] - \frac{1}{|\mathcal{I}_k^y|}\sum_{i\in\mathcal{I}_k^y}\ell(f_\theta^S(x_i),y_i) \qquad (7)\\
&= \mathbb{E}_z[\ell((d_\delta\circ v_\gamma\circ k_\beta\circ z_\alpha)(x),y)|z\in\mathcal{C}_k^y] - \frac{1}{|\mathcal{I}_k^y|}\sum_{i\in\mathcal{I}_k^y}\ell((d_\delta\circ v_\gamma\circ k_\beta\circ z_\alpha)(x_i),y_i) \\
&= \ell((d_\delta\circ v_\gamma)(\mathcal{K}_k),y) - \ell((d_\delta\circ v_\gamma)(\mathcal{K}_k),y) = 0.
\end{aligned}$$

For the case of $f = f_\theta^S$, by combining equation (5) and (7), it holds that any $\delta > 0$, with probability at least $1-\delta$,

$$\begin{aligned}
&\mathbb{E}_{z,\epsilon}[\ell(f_\theta^S(g_\epsilon(x)),y)] - \frac{1}{n}\sum_{i=1}^{n}\ell(f_\theta^S(x_i),y_i) \\
&\leq c\sqrt{\frac{2\ln(2e/\delta)}{n}} + \frac{1}{n}\sum_{y,k}|\mathcal{I}_k^y|\mathbb{E}_{x\sim\mathcal{D}_{x|y},\epsilon}\left[\ell(f_\theta^S(g_\epsilon(x)),y) - \ell(f_\theta^S(x),y)|x\in\mathcal{C}_k\right]
\end{aligned}$$

Moreover, for the case of $f = f_\theta^S$, if $x\in\mathcal{C}_k \Rightarrow g_\epsilon(x)\in\mathcal{C}_k$ with probability one, then

$$\begin{aligned}
\mathbb{E}_{x\sim\mathcal{D}_{x|y},\epsilon}\left[\ell(f_\theta^S(g_\epsilon(x)),y) - \ell(f_\theta^S(x),y)|x\in\mathcal{C}_k\right] &= \ell((d_\delta\circ v_\gamma)(\mathcal{K}_k),y) - \ell((d_\delta\circ v_\gamma)(\mathcal{K}_k),y) \\
&= 0.
\end{aligned}$$

□

B. Why Do Random Down-Projection Matrices Help?

In addition to the flexibility in reducing the term $\frac{1}{n}\sum_{i=1}^{n}\ell(f(x_i),y_i)$ when having multiple codebooks, the benefit from random projections in terms of Theorem 3.1 is that this projection can reduce the term of $\mathcal{B}_{g_\epsilon}(f)$. For example, consider $z+v$ where z is an embedding and v is some perturbation of z. When computing a projection with a random matrix R by $R(z+v)$ and v is in the null space of R, then $R(z+v) = Rz$ and hence this perturbation has no effect. Since R is for down-sampling, there is a large null space for R. In other words, if the distribution shift results in v such that Rv is small, $\mathcal{B}_{g_\epsilon}(f)$ is small.

C. Implementation Details.

Below we summarize all relevant implementation details.[1]

Model Architecture. Our experiments build upon the following model architecture. **Encoder:** We use the following publicly available backbones:

1. ResNet50 backbone, pre-trained supervised on ImageNet (He et al., 2016; Russakovsky et al., 2015): We used the weights from PyTorchs `torchvision.models` package, specifically the weights under identifier `ResNet50_Weights.IMAGENET1K_V2` [2]. The backbone yields an embedding dimension of $m_z = 2048$.

[1] https://github.com/ftraeuble/experiments_discrete_key_value_bottleneck.
[2] https://pytorch.org/vision/stable/models.html

2. ViT-B/32 backbone, CLIP-pre-trained (Radford et al., 2021): We used the ViT B/32 model that can be easily obtained via OpenAI's official github repository `clip` python package [3]. The backbone yields an embedding dimension of $m_z = 512$.

3. ResNet50w2 with twice the standard wideness, SwAV self-supervised pre-trained (Caron et al., 2020): We use the official pytorch model that can be loaded with PyTorch using the command `torch.hub.load('facebookresearch/swav:main', 'resnet50w2')`. The backbone yields an embedding dimension of $m_z = 2048$.

4. ResNet50, DINO self-supervised pre-trained (Caron et al., 2021): We use the official pytorch model that can be loaded with PyTorch using the command `torch.hub.load('facebookresearch/dino:main', 'dino_resnet50')`. The backbone yields an embedding dimension of $m_z = 1024$.

5. As a fifth model, we present results using a ConvMixer pre-trained on a downsampled (32x32) Imagenet training set. As we wanted to directly compare against SDMLPs performance on the same class-incremental learning setting, we use their ConvMixer embeddings evaluated on the entire Imagenet and CIFAR10 dataset, which can be downloaded from the associated github repository.[4] The backbone yields an embedding dimension of $m_z = 256$.

To cover a broad range of feature embeddings, we extracted the spatial continuous representation after the second last layer of residual blocks in the case of the two self-supervised pre-trained backbones and applied adaptive average pooling. The inferred continuous-valued representations cover a broad range of embedding dimensionalities depending on the backbone. In all our experiments we obtained the C embedding heads for each key-value codebook by simply down-projecting this vector using C random projection matrices into C heads of dimension d_{key}. **Key-Value Bottleneck:** The key-value bottleneck consists of 256 key-value codebooks that have each 4096 key-value pairs per codebook. Keys are of the same dimension as the embedding heads (d_{key}), and we chose the value codes to be of the same size as the classes to predict, i.e. $d_{value} = 10$. **Decoder:** All C fetched value codes are element-wise average-pooled at the end and the resulting tensor is passed to a softmax layer that predicts the class label. For simplicity of exposition, we assume that the encoder representation does not have spatial dimensions; nevertheless, the extension to spatial feature maps is straightforward.

Initialization of Keys. To initialize the keys, we build upon the exponential moving average updates (EMA) implemented in the pip package `vector-quantize-pytorch`[5] as introduced in (Razavi et al., 2019; Van Den Oord et al., 2017). For a given sequence of $t = 1, ..., T$ mini-batches, we compute the moving averages of the codes (=key) positions $k_i^{(t)}$ and counts $N_i^{(t)}$ as follows for the c-th codebook

$$N_i^{(t)} := \gamma N_i^{(t-1)} + n_i^{(t)}(1-\gamma), \tag{8}$$

$$m_i^{(t)} := \gamma m_i^{(t-1)} + \sum_j^{n_i^{(t)}} E^c(x)_{i,j}^{(t)}(1-\gamma), \tag{9}$$

$$k_i^{(t)} := \frac{m_i^{(t)}}{N_i^{(t)}} \tag{10}$$

where $E^c(x)^{(t)}_{i,j=1,\dots,n_i^{(t)}}$ are the $n_i^{(t)}$ (head) embeddings of the set of observations in the mini-batch that attach to the i-th key. We use a decay factor of $\gamma = 0.95$. Moreover, we expire codes if they are not being adequately utilized. If the cluster size corresponding to a key (i.e., the moving average of the number of encodings that attach to a key) is below a threshold, we reinitialize the said key to a randomly selected encoding. This threshold scales as: $0.1 \cdot \text{batch-size} \cdot h \cdot w / \text{num-pairs}$ with h and w the spatial feature map dimensions in case of spatial pooling after the bottleneck. Keys are initialized at random encoder feature positions from the first batch. Our experiments apply spatial pooling before the bottleneck such that $h = w = 1$.

[3] `https://github.com/openai/CLIP`
[4] `https://github.com/anon8371/AnonPaper1`
[5] `https://github.com/lucidrains/vector-quantize-pytorch`

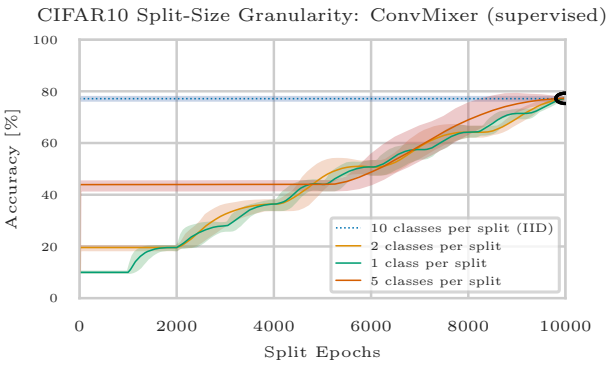

Figure 6. Analysing the effect of class-granularity during class incremental learning. In line with our theory, we observe *across all possible class-split sizes* the appealing learning property that the bottleneck yields the same performance at the end of training.

Training. After initializing the keys for 10 epochs on the key-initialization dataset (1 epoch on the Imagenet datasets), we train the model on the class-incremental CIFAR10 task with the SGD PyTorch optimizer without any weight decay or momentum and a learning rate of $lr = 0.3$ for the bottleneck and $lr = 0.001$ for the linear probe. We used a label smoothing parameter of 0.1 though we did not find its value to be sensitive to the result. We use no learning rate schedule. We used a batch size of 256 during key initialization and continual learning.[6]

D. Additional Experiments

D.1. Granularity During Class-Incremental Learning.

In addition to the class-incremental learning scenario discussed in the main text, we experimented with additional class-incremental curricula of different split granularities. Specifically, we also tested the same model's performance (with the ConvMixer backbone) not only on 5 phases with 2 classes each but also on the more extreme case of 10 phases with a single class each and two phases with 5 classes each. We show these results in Figure 6, further supporting our finding from the main experiment that the proposed model is robust against various class-incremental curricula and approaches the same performance at the end of training.

D.2. Additional Experiments on ConvMixer Backbone.

The authors in (Bricken et al., 2023) report two further methods with higher performance but their "SDMLP+EWC" results does not apply as it requires task boundaries and the FlyModel from Shen et al. (2021) was not trained on the batched 2000 epochs streaming setup. However, we also attempted to implement the FlyModel that proposes a solution for continual learning inspired by the neural circuit in the fruit fly olfactory system. Similar to us, the FlyModel also relies on a frozen backbone but then projects this embedding via sparse binary projections (which are likewise fixed) into an extremely high-dimensional space reflecting more than 10,000 so-called "Kenyon cells". It then instantiates num-class "MBON cells" that are associative layers, fed from this high-dimensional representation. The FlyModel is trained such that learning a particular class updates the "MBON" associative layer of this class and was primarily tested under less challenging task-incremental settings by the authors. As mentioned before, the model reported in (Bricken et al., 2023) was not trained for 2000 epochs per split. Instead, it was only updated over a single epoch simply assuming no further catastrophic forgetting in the batched streaming setting we are investigating (the FlyModel is not trained using back-propagation). In order to make the comparison as fair as possible, we trained the FlyModel for 2000 epochs per split building upon the implementation in (Bricken et al., 2023). Following Bricken et al. (2023), we applied the following parameters that were reported to work best for them on this task and this backbone, specifically a learning rate of 0.005, 10,000 Kenyon cells and the number of projection neuron connections to be 32. We also set $\alpha = 0$ to minimize catastrophic forgetting as described by Shen et al. (2021). After rerunning this model for 5 random seeds each we obtained the results presented in Figure 7. As we can see, the model learns new classes at the beginning of a new split but quickly drops to some lower-level performance until the end

[6]To the best of our knowledge, all software and assets we use are open source and under MIT, Apache or Creative Commons Licenses.

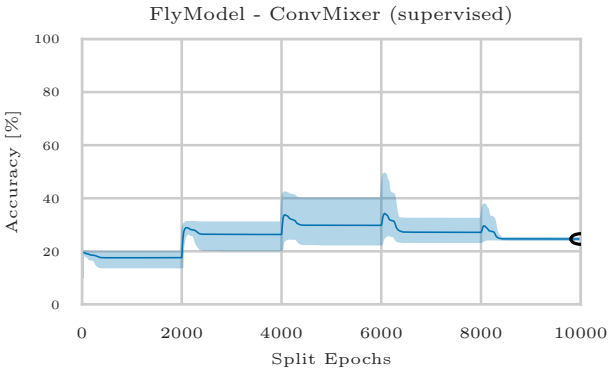

Figure 7. Applying the FlyModel from (Shen et al., 2021) to the CIFAR10 learning setup with each split being repeated 2000 epochs. The model with hyperparameters from (Bricken et al., 2023) learns new classes at the beginning of a new split but quickly drops to some lower-level performance until the end of the phase. Eventually, the model cannot integrate any new information and remains at a very low level.

of the phase. Eventually, the model cannot integrate any new information and remains at a very low level, which indicates strong catastrophic forgetting in the batched streaming setting.

D.3. Additional Baseline: Ridge Regression on Frozen Backbone Features

We have also tested a further baseline method by directly performing ridge regression with the frozen feature encoder. Specifically, we were rerunning the CIFAR10 experiment on all five backbones using two different heads (Linear Probe and as well as the MLP head from Figure 3). In contrast to these baselines already shown in the main paper, we now applied weight decay on the head parameters across four orders of magnitude with $w_d \in [1e-2, 1e-4, 1e-6]$. We repeated three runs with different random seeds each. In total, we trained 90 additional models for this experiment. As illustrated in Figure 8 in the case of the Linear Probe and Figure 9 in the case of the MLP head, the learned networks were unable to effectively integrate novel class information using this additional form of ridge regularization.

D.4. Class-Incremental Learning on Larger Datasets.

To further test the future scalability of our method, we have run additional experiments on two larger datasets with many more classes. Importantly, we were using the very same architecture as for the CIFAR10 experiments:

1. CIFAR-100 (see Figure 10): Using 50 splits, each consisting of 2 classes, we tested our model on the ResNet-50 and CLIP-Backbone architectures. Our model achieved final accuracies of 54.7% and 64% respectively, compared to the 43% achieved by SDMLP on the same task (Bricken et al., 2023).

2. ImageNet (see Figure 11): We evaluated our model on 500 splits, each comprising 2 classes, using the CLIP encoder. The final accuracy reached 49.9% for the iid setting and 49.0% for the class-incremental setting. To the best of our knowledge, no other method has been tested on such a large-scale class-incremental learning setting, encompassing 1000 classes and 500 splits.

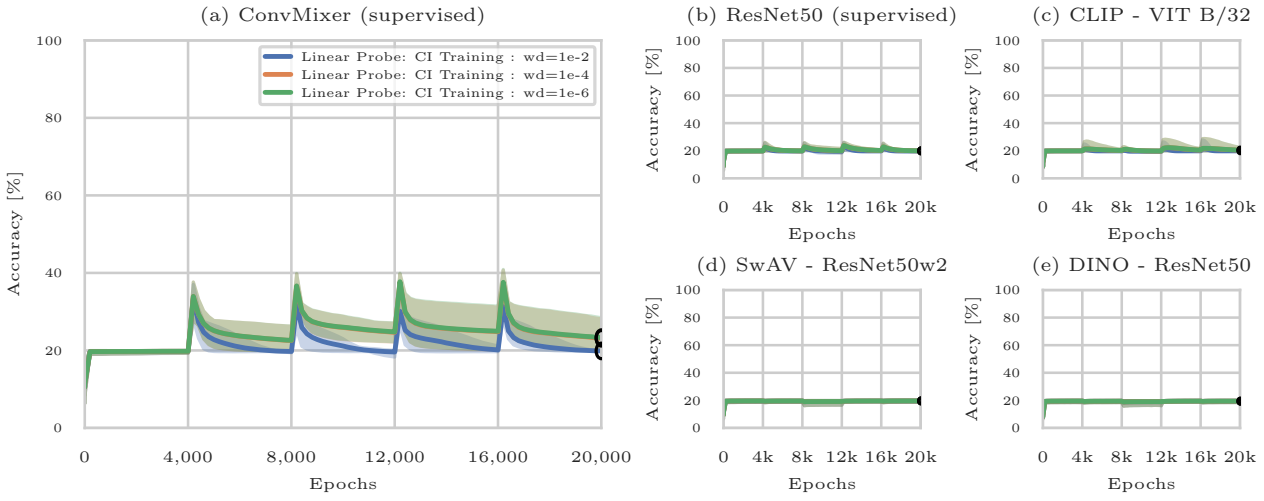

Figure 8. Additional baseline experiments: Rerunning the CIFAR10 experiment on all five backbones using the same Linear Probe as in Figure 3. In contrast to these baselines already shown in the main paper, we now applied weight decay on the head parameters across four orders of magnitude with $w_d \in [10^{-2}, 10^{-4}, 10^{-6}]$. The learned models are unable to effectively integrate novel class information using this additional form of ridge regularization.

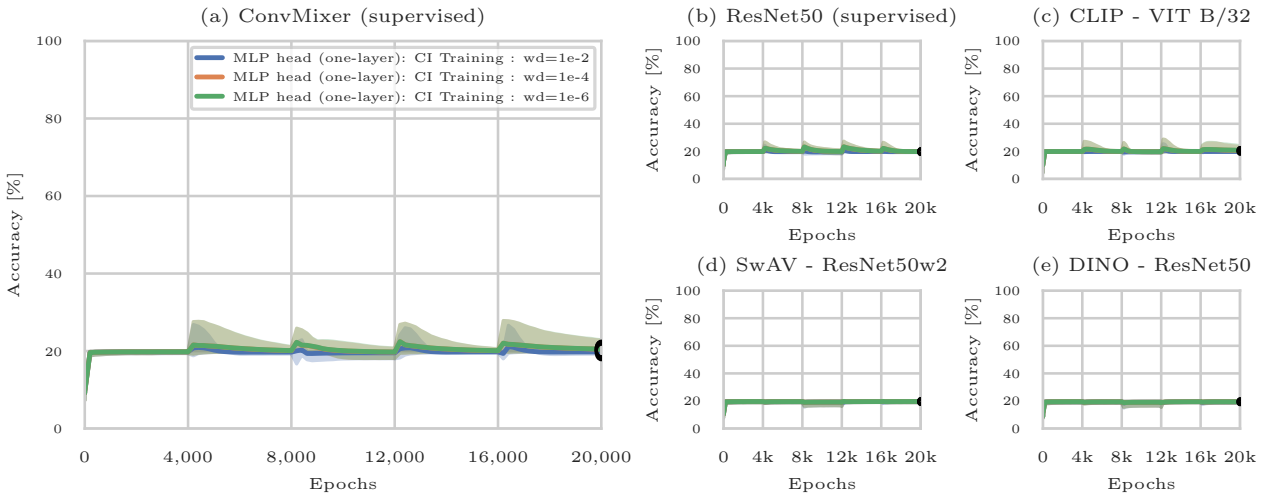

Figure 9. Additional baseline experiments: Rerunning the CIFAR10 experiment on all five backbones using the same MLP head as in Figure 3. In contrast to these baselines already shown in the main paper, we now applied weight decay on the head parameters across four orders of magnitude with $w_d \in [10^{-2}, 10^{-4}, 10^{-6}]$. The learned models are unable to effectively integrate novel class information using this additional form of ridge regularization.

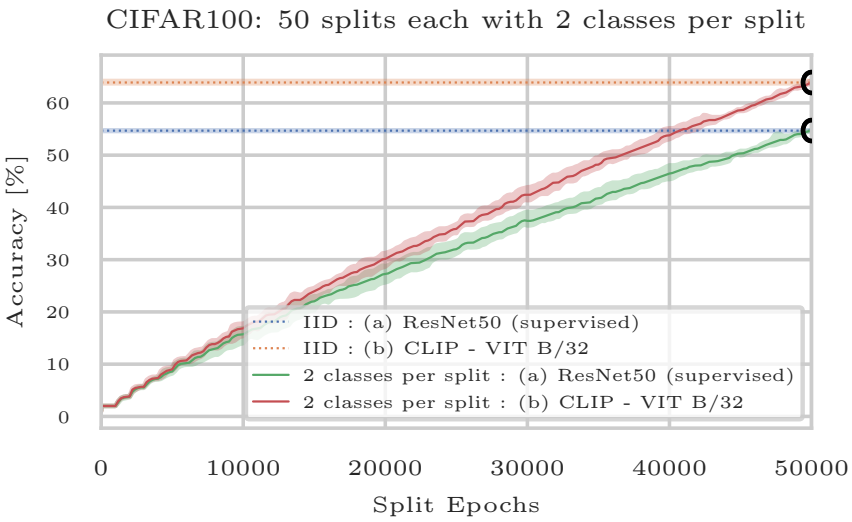

Figure 10. Additional class-incremental learning experiment on CIFAR100: Using 50 splits, each consisting of 2 classes, we tested our model on the ResNet-50 and CLIP-Backbone architectures. Our model achieved final accuracies of 54.7% and 64% respectively, compared to the 43% achieved by SDMLP on the same task (Bricken et al., 2023). Keys were initialized on the CIFAR10 dataset for this experiment.

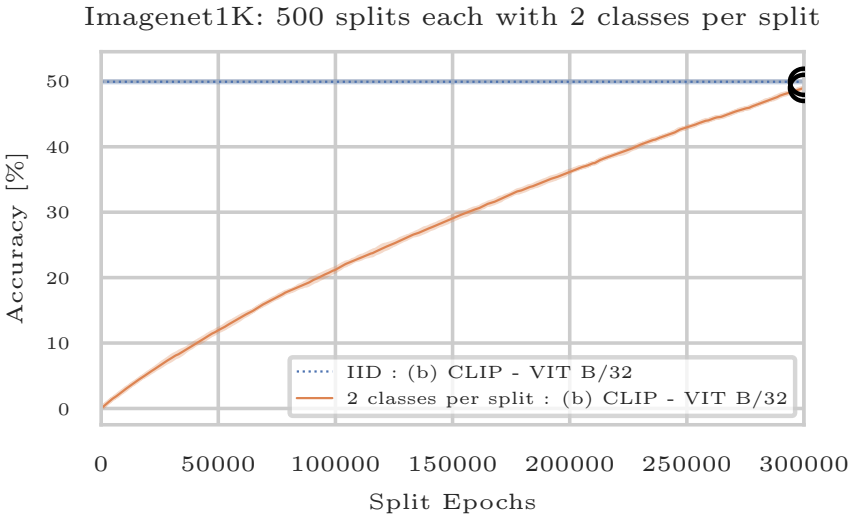

Figure 11. Additional class-incremental learning experiment on ImageNet: We evaluated our model on 500 splits, each comprising 2 classes, using the CLIP encoder. The final accuracy reached 49.9% for the iid setting and 49.0% for the class-incremental setting. To the best of our knowledge, no other method has been tested on such a large-scale class-incremental learning setting, encompassing 1000 classes and 500 splits. Keys were initialized on the CIFAR100 dataset for this experiment.

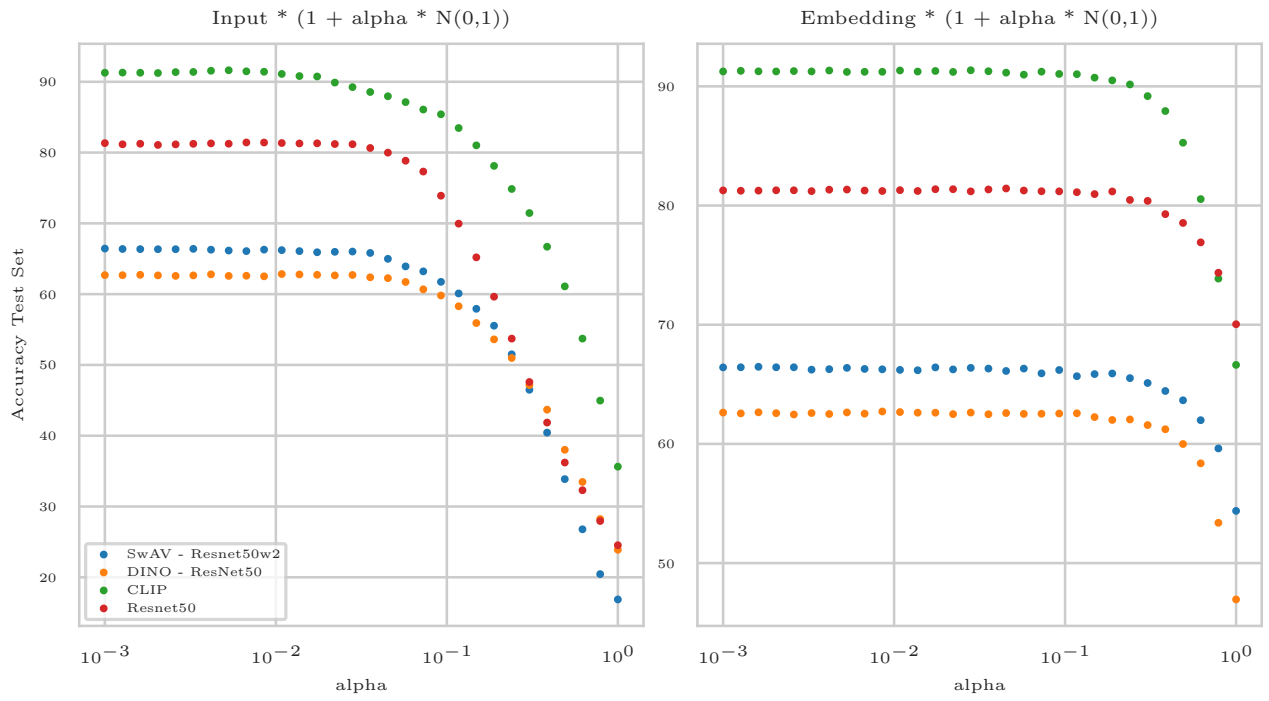

Figure 12. We have conducted additional experiments to assess the models' robustness. Specifically, we evaluated the performance of the models analyzed in Figures 8-11, which employ different backbones, when subjected to additive Gaussian noise spanning three orders of magnitude. We applied the noise either to the original input pixels (left) or the backbone embeddings (right). Results indicate that the model's performance remains largely unaffected by the noise and only starts to deteriorate substantially when the noise level exceeds $\alpha = 0.1$, at which point the image becomes nearly unrecognizable upon visual inspection. From a theoretical per

E. Additional Analysis

E.1. Robustness via Discrete Bottleneck.

The discretization of the bottleneck confers additional robustness, which is a significant attribute of our model. To illustrate this property, we have conducted additional experiments to assess the models' robustness. Specifically, we evaluated the performance of the models analyzed in Figures 13 to 17, which employ different backbones, when subjected to additive Gaussian noise spanning three orders of magnitude. We applied the noise either to the original input pixels or the backbone embeddings. Our results which are shown in Figure 12, indicate that the model's performance remains largely unaffected by the noise and only starts to deteriorate substantially when the noise level exceeds $\alpha = 0.1$, at which point the image becomes nearly unrecognizable upon visual inspection.

E.2. Value Codes

Next, we want to provide additional analysis on the similarity of the learned values when training iid versus class-incrementally. To answer how much the value codes differ depending on the training order (everything else being the same), we computed the relative mean absolute difference between values learned iid and values learned class-incrementally in the case of the five analyzed models in the previous section, formally

$$\frac{\sum |values_{CI} - values_{IID}|}{\sum |values_{CI}|}. \tag{11}$$

As we can see from Table 1, the analysis suggests that the learned values are indeed fairly similar, albeit not identical, explaining the almost identical final performance.

Backbone	Rel. mean abs. distance
Resnet50	2.19%
CLIP	2.09%
DINO	2.2%
SwAV	2.2%
ConvMixer	3.07%

Table 1. Relative mean absolute distance between values trained iid and class-incremental for the five different backbones from the base experiment.

E.3. Key Codes

In this section, we analyze key codes used in the proposed model to better understand how they cover the embedding manifold of the data. We investigate three questions:

1. Do key codes broadly cover the embedding manifold with respect to the key initialization dataset?
2. How does key initialization on another dataset affect coverage on the embedding manifold under the target dataset?
3. Can we quantify the amount of key-value pair sharing among training samples?

To answer these questions, we inspect the key codes for each backbone discussed in Section 5, analyzing the model per backbone with the same fixed random seed trained under the class-incremental training setting. Results are shown in Figures 13 to 17.

We begin by inspecting the key code space of the first codebook under the supervised ResNet50 backbone in Figure 13 (left plot). The keys of this model were initialized on the unlabelled CIFAR100 dataset. When applying the UMAP dimensionality reduction on the input heads of 10k CIFAR100 training samples, we can see that the 4096 key codes of this codebook (black dots) broadly cover the entire data manifold (light blue dots). This verifies that the EMA method is working as expected. When repeating the same procedure for the test set of the target distribution, here CIFAR10, we observe that the fixed keys from before are still covering the entire data domain of CIFAR10, see Figure 13 (middle plot). Compared to the distribution on the key initialization dataset, the coverage is more sparse with several concentrations of multiple clusters of keys. We hypothesize that these keys might have specialized on certain features present in CIFAR100 but not present in CIFAR10. At the same time, we observe keys across all CIFAR10 class domains (indicated by color) even though the same class labels are not present in CIFAR100. We observe this behaviour across all backbones: Interestingly, we qualitatively observe a slightly reduced clustering in the case of the lower-level SSL feature embeddings in Figure 15 and Figure 16 over the others. In our ablation experiments, we only observed a slight drop in performance when initializing the keys on the much higher-resolution ImageNet dataset, which can be partially explained with the analysis under the ConvMixer model, that was initialized on a 32x32 downsampled Imagenet variant (see Figure 17). First, the EMA approach still yields a broad coverage of the much more diverse Imagenet data manifold without any change in EMA parameters. Second, the keys are covering the CIFAR10 domain sparsely but there are still at least a few keys present close to each domain. From our discussed results we can thereby conclude that the bottleneck can still effectively learn to generalize along all classes even though the keys might not be optimally distributed. Finally, we tracked the usage of all key-value pairs across all codebooks over the course of the training class incrementally on the target dataset; see Figure 13 (right plot). Note that the y-axis is scaling in counts of thousands. In the case of the ResNet50 backbone, we observe that the majority of key-value pairs were fetched between 10,000 and 150,000 times over the course of training. As we repeat each episode for 2000 epochs, each key that is snapped to by a particular training image contributes a utilization count of 2000 to this key. Interestingly, there is also a long tail of a few keys that are used very often and about 28% of keys that are never used, which suggests that the bottleneck did not distil the information during train time in all available key-value codes. This behaviour can be seen similarly among all other backbones, too.

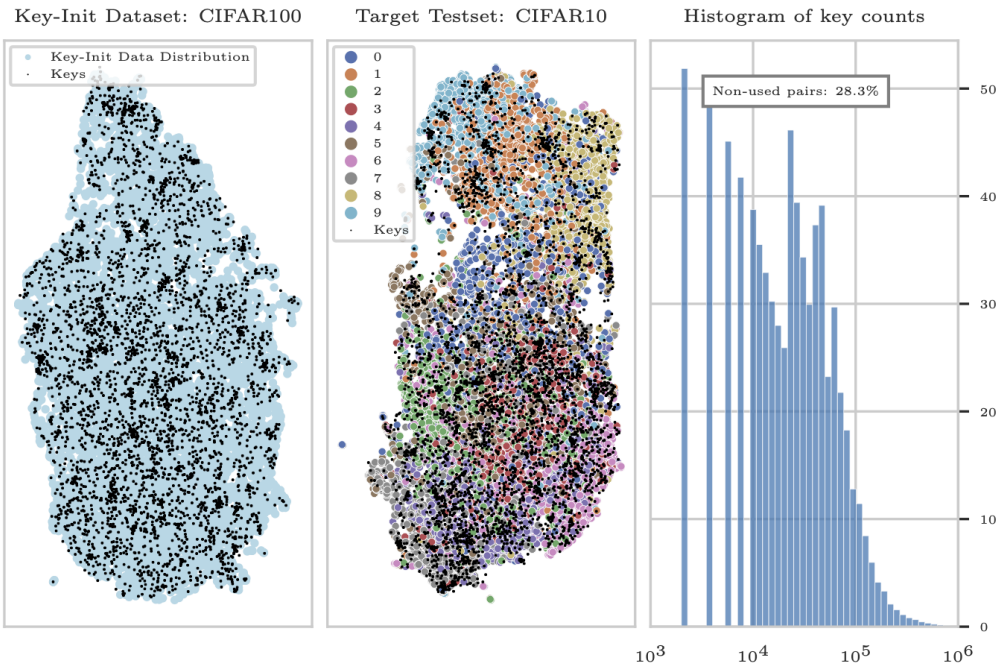

Figure 13. Analysing the **key codes** on the **ResNet50 backbone.** Left Plot: UMAP of the codebook embedding heads from the key initialization dataset (light blue) with the discrete key codes (black dots) as obtained by EMA. Keys are broadly covering the data manifold. Middle Plot. The key code distribution on the target test dataset. Keys exhibit increased clustering, which reflects the smaller feature diversity present in CIFAR10. Right Plot: Key code utilization across all codebooks over the course of class-incremental training (2000 epochs). The majority of keys is being shared among many different input samples, while the bottleneck retains some unused capacity.

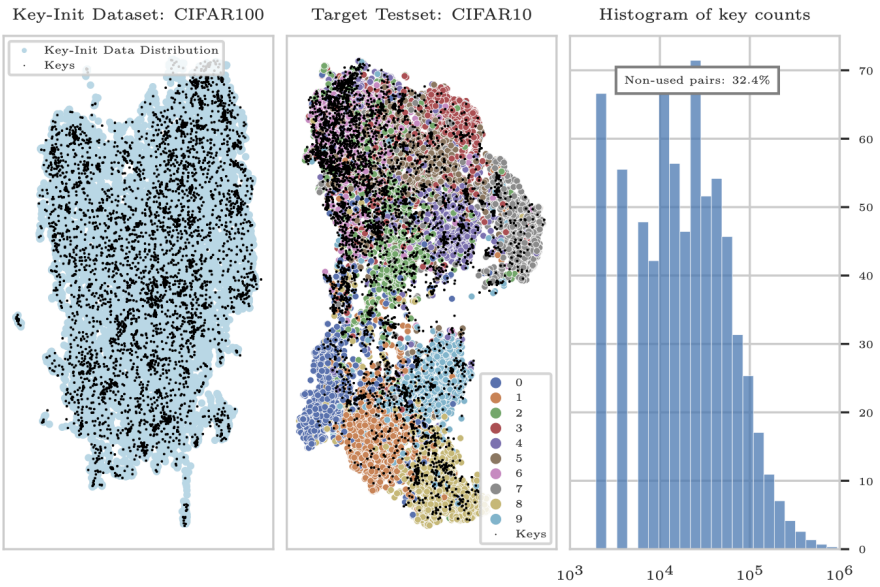

Figure 14. Analysing the **key codes** on the **CLIP-pretrained ViT backbone**. Left: UMAP of the codebook embedding heads from the key initialization dataset (light blue) with the discrete key codes (black dots) as obtained by EMA. Keys are broadly covering the data manifold. Middle. The key code distribution on the target test dataset. Although the coverage appears more clustered, the keys still cover the majority of the target test set, which reflects the smaller feature diversity present in CIFAR10. Right: Key code utilization across all codebooks over the course of class-incremental training (2000 epochs). The majority of keys is being shared among many different input samples, while the bottleneck retains some unused capacity.

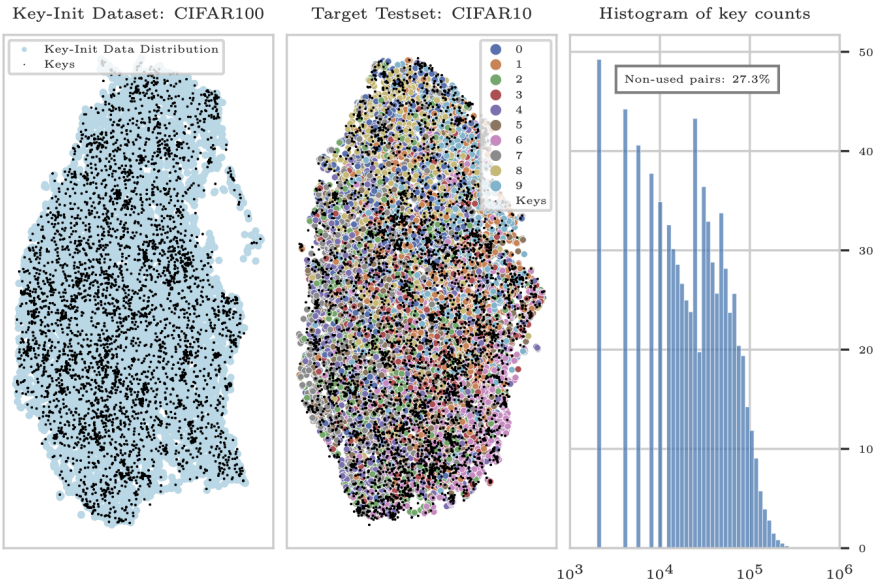

Figure 15. Analysing the **key codes** on the **DINO-pretrained backbone**. Left: UMAP of the codebook embedding heads from the key initialization dataset (light blue) with the discrete key codes (black dots) as obtained by EMA. Keys are broadly covering the data manifold. Middle. The key code distribution on the target test dataset. Although the coverage appears more clustered, the keys still cover the majority of the target test set, which reflects the smaller feature diversity present in CIFAR10. Right: Key code utilization across all codebooks over the course of class-incremental training (2000 epochs). The majority of keys is being shared among many different input samples, while the bottleneck retains some unused capacity.

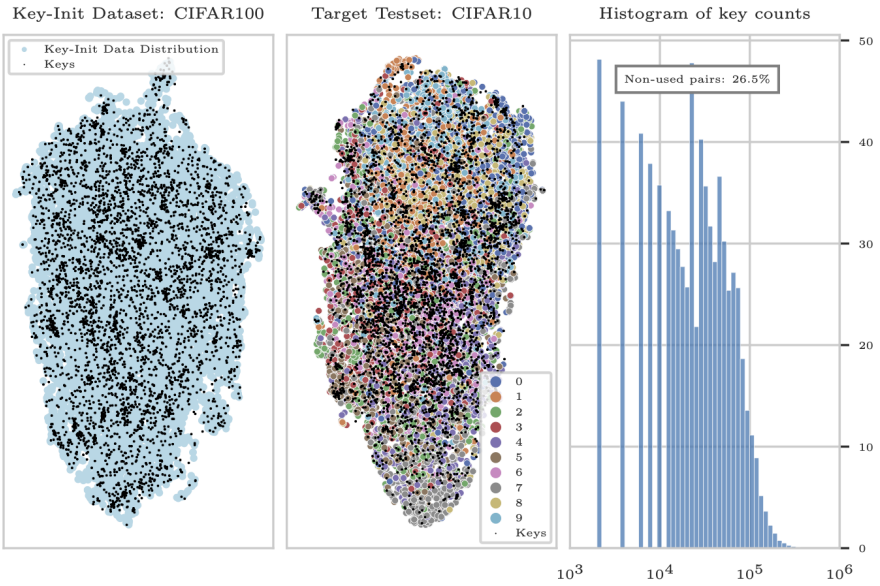

Figure 16. Analysing the **key codes** on the **SwAV-pretrained backbone.** Left: UMAP of the codebook embedding heads from the key initialization dataset (light blue) with the discrete key codes (black dots) as obtained by EMA. Keys are broadly covering the data manifold. Middle. The key code distribution on the target test dataset. Keys exhibit increased clustering, which reflects the smaller feature diversity present in CIFAR10. Right: Key code utilization across all codebooks over the course of class-incremental training (2000 epochs). The majority of keys is being shared among many input samples, while the bottleneck retains some unused capacity.

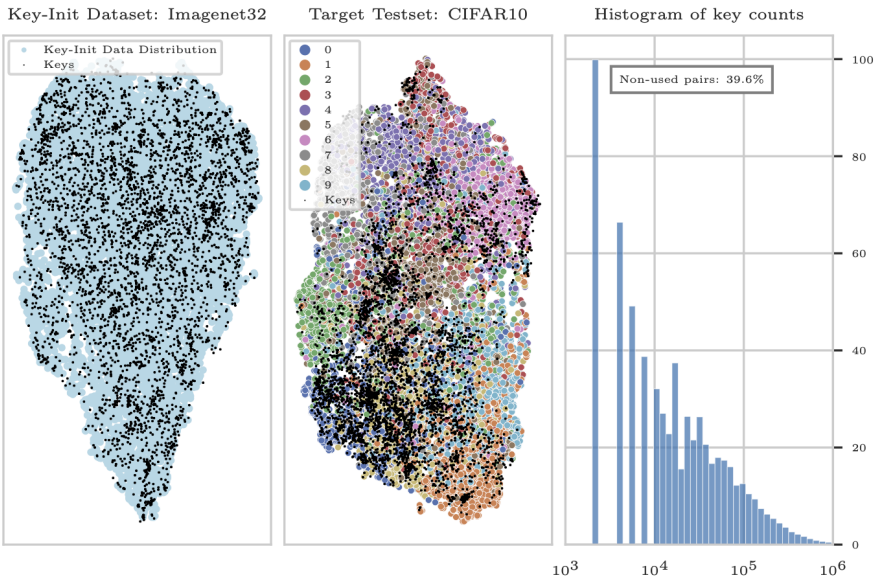

Figure 17. Analysing the **key codes** on the **ConvMixer backbone** from (Bricken et al., 2023). Left: UMAP of the codebook embedding heads from the key initialization dataset (light blue) with the discrete key codes (black dots) as obtained by EMA. Keys are broadly covering the data manifold. Middle. The key code distribution on the target test dataset. Keys exhibit increased clustering, which reflects the smaller feature diversity present in CIFAR10. Right: Key code utilization across all codebooks over the course of class-incremental training (2000 epochs). The majority of keys is being shared among many input samples, while the bottleneck retains some unused capacity.