Toward Sustainable Continual Learning: Detection and Knowledge Repurposing of Similar Tasks

Sijia Wang\textsuperscript{1}, Yoojin Choi\textsuperscript{2}, Junya Chen\textsuperscript{1}, Mostafa El-Khamy\textsuperscript{2}, and Ricardo Henao\textsuperscript{1}

\textsuperscript{1}Duke University\hspace{1cm}\textsuperscript{2}SoC R&D Samsung Semiconductor Inc.\hspace{1cm}sijia.wang@duke.edu

Abstract

Most existing works on continual learning (CL) focus on overcoming the catastrophic forgetting (CF) problem, with dynamic models and replay methods performing exceptionally well. However, since current works tend to assume exclusivity or dissimilarity among learning tasks, these methods require constantly accumulating task-specific knowledge in memory for each task. This results in the eventual prohibitive expansion of the knowledge repository if we consider learning from a long sequence of tasks. In this work, we introduce a paradigm where the continual learner gets a sequence of mixed similar and dissimilar tasks. We propose a new continual learning framework that uses a task similarity detection function that does not require additional learning, with which we analyze whether there is a specific task in the past that is similar to the current task. We can then reuse previous task knowledge to slow down parameter expansion, ensuring that the CL system expands the knowledge repository sublinearly to the number of learned tasks. Our experiments show that the proposed framework performs competitively on widely used computer vision benchmarks such as CIFAR10, CIFAR100, and EMNIST.

1 Introduction

Human intelligence is distinguished by the ability to learn new tasks over time while remembering how to perform previously experienced tasks. Continual learning (CL), an advanced machine learning paradigm requiring intelligent agents to continuously learn new knowledge while trying not to forget past knowledge, has a pivotal role in machines imitating human-level intelligence \cite{1}. The main problem in continual learning is catastrophic forgetting (CF) of previous knowledge when new tasks, observed over time, are incorporated to the model via training or (parameter) expansion.

In general, continual learning methods can be categorized into three major groups based on: (i) regularization, (ii) replay, and (iii) dynamic (expansion) models. Approaches based on regularization \cite{2-4} alleviate forgetting by constraining the updates of parameters that are important for previous tasks, by adding a regularization term to the loss function penalizing changes between parameters for current and previous tasks. While this type of approach excels at keeping a low memory footprint, the ability to remember previous knowledge eventually declines, especially, in scenarios with long sequences of tasks \cite{5}. Methods based on replay keep a memory bank to store either a subset of exemplars to represent each class and task \cite{6-8}, or generative models for pseudo-replay \cite{9, 10}. Approaches based on dynamic models \cite{11-15} allow the architecture (or components of it) to expand over time. Specifically, \cite{13, 16} leverage modular networks and study a practical continual learning scenario with 100-task long sequences. Nevertheless, these parameter expansion approaches face the challenges of keeping model growth under control (ideally sublinear). This problem associated with parameter expansion in CL systems is critical and requires further attention.

Preprint. Under review.
This paper discusses CL under a task continual learning (TCL) setting, i.e., that in which data arrives sequentially in groups of tasks. For works under this scenario [17, 18], the assumption is usually that once a new task is presented, all of its data becomes readily available for batch (offline) training. In this setting, a task is defined as an individual training phase with a new collection of data that belongs to a new (never seen) group of classes, or in general, a new domain. Further, TCL also (implicitly) requires a task identifier during training. However, in practice, when the model has seen enough tasks, a newly arriving batch of data becomes increasingly likely to belong to the same group of classes or domain of a previously seen task. Importantly, most existing works on TCL fail to acknowledge this possibility. Moreover and in general, the task definition or identifier may not be available during training, e.g., the model does not have access to the task description due to (user) privacy concerns. In such case mostly concerning dynamic models, the system has to treat every task as new, thus constantly learning new sets of parameters regardless of task similarity or overlap. This clearly constitutes a suboptimal use of resources (predominantly memory), especially as the number of tasks experienced by the CL system grows.

This study investigates the aforementioned scenario and makes it an endeavor to create a memory-efficient CL system which though focused on image classification tasks, is general and in principle can be readily used toward other applications or data modality settings. We provide a solution for dynamic models to identify similar tasks when no task identifier is provided during the training phase. To the best of our knowledge, the only work that also discusses the learning of a continual learning system with mixed similar and dissimilar tasks is [19], which proposes a task similarity function to identify previously seen similar tasks, which requires training a reference model every time a new task becomes available. Alternatively, in this work, we identify similar tasks without the need for training a new model, by leveraging a task similarity metric, which in practice results in high task similarity identification accuracy. We also discuss memory usage under challenging scenarios where longer, more realistic, sequences of more than 20 tasks are used.

To summarize, our contributions are listed below:

• We propose a new framework for an under-explored and yet practical TCL setting in which we seek to learn a sequence of mixed similar and dissimilar tasks, while preventing (catastrophic) forgetting and repurposing task-specific parameters from a previously seen similar task, thus slowing down parameter expansion.
• The proposed TCL framework is characterized by a task similarity detection module that determines, without additional learning, whether the CL system can reuse the task-specific parameters of the model for a previous task or needs to instantiate new ones.
• Our task similarity detection module shows remarkable performance on widely used computer vision benchmarks, such as CIFAR10 [20], CIFAR100 [20], EMNIST [21], from which we create sequences of 10 to 100 tasks.

2 Related Work
TCL in Practical Scenarios

Task continual learning (TCL) being an intuitive imitation of human learning process constitutes one of the most studied scenarios in CL. Though TCL systems have achieved impressive performance [22, 23], previous works have mainly focused on circumventing the problems associated with CF. Historically, task sequences have been restricted to no more than 10 tasks and strong assumptions have been imposed so that all the tasks in the sequence are unique and classes among tasks are disjoint [10, 24]. The authors of [25] rightly argue that currently discussed CL settings are oversimplified, and more general and practical CL forms should be discussed to advance the field. It is not until recently that solutions have been proposed for longer sequences and more practical CL scenarios. Particularly, [19] proposed CAT, which learns from a sequence of mixed similar and dissimilar tasks, thus enabling knowledge transfer between future and past tasks detected to be similar. To characterize the tasks, a set of task-specific masks, i.e., binary matrices indicating which parameters are important for a given task [22], are trained along other model parameters. Specifically, these masks are activated and parameters associated to them are finetuned once the current task is identified as “similar” by a task similarity function, or otherwise held fixed by the masking parameters to protect them from changing, hence preventing CF. Alternatively, [15] introduces a modular network, which is composed of neural modules that are potentially shared with related tasks. Each task is optimized by selecting a set of modules that are either freshly trained on
We consider the TCL scenario for image classification tasks, where we seek to incrementally learn global parameters that can be adapted to new tasks in an efficient manner. Instead of retraining the model from scratch for every new task, we do not have to instantiate completely different sets of parameters for every unique task, but rather transfer parameters from previously learned tasks. This approach can significantly reduce the memory footprint, required for new tasks. Specifically, the feature transformation involves two steps: 1) Learning a backbone model that is task-agnostic and 2) Implementing small convolutional transformations on the learned backbone to adapt it to new tasks.

In our setting, only $W$ is learned for task $T$, as a means to reduce the parameter count, and thus the memory footprint, required for new tasks. The efficient feature transformation (EFT) framework in [30] proposed that instead of finetuning all the parameters in a well-trained (pretrained) model, one can instead partition the network into a (global) backbone and task-specific feature transformation layers.

3 Background

3.1 Problem Setting

We consider the TCL scenario for image classification tasks, where we seek to incrementally learn a sequence of tasks $T_1, T_2, \ldots, T_{t-1}$ and denote the collection of tasks currently learned as $T = \{T_1, \ldots, T_{t-1}\}$. The underlying assumption is that, as the number of tasks in $T$ grows, the current task $T_t$ will eventually have a corresponding similar task $T_{sim} \in T$. Let the set of all dissimilar tasks be $T_{dis} = T \setminus T_{sim}$. We define similar and dissimilar tasks as follows.

**Similar and Dissimilar tasks:** Consider two tasks $A$ and $B$, which are represented by datasets $D_A = \{x_i^A, y_i^A\}_{i=1}^{n_A}$ and $D_B = \{x_i^B, y_i^B\}_{i=1}^{n_B}$, where $y_i^A \in \{Y^A\}$ and $y_i^B \in \{Y^B\}$. If predictors (e.g., images) in $\{x_i^A\}_{i=1}^{n_A} \sim P$ and $\{x_i^B\}_{i=1}^{n_B} \sim P$, and labels $\{Y^A\} = \{Y^B\}$, indicating that data from $A$ and $B$ belong to the same group of classes, and their predictors are drawn from the same distribution $P$, then we say $A$ and $B$ are similar tasks, otherwise, $A$ and $B$ are deemed dissimilar. Notably, when both tasks share the same distribution $P$, but have different label spaces, they are considered dissimilar.

The main objective in this work is to identify $T_{sim}$ among $T$ without training a new model (or learning parameters) for $T_t$, by leveraging a task similarity identification function, that will enable the system to reuse the parameters of $T_{sim}$ when identification of a previously seen task is successful. Alternatively, the system will instantiate new parameters for the dissimilar task. As a result, the system will attempt to learn parameters for the set of unique tasks, which in a long sequence of tasks is assumed to be smaller than the sequence length. In practice, in order to handle memory efficiently, we do not have to instantiate completely different sets of parameters for every unique task, but rather define global and task-specific parameters as a means to further control model growth. For this purpose, we leverage the efficient feature transformations for convolutional models described below.

3.2 Task-Specific Adaptation via Feature Transformation Techniques

The efficient feature transformation (EFT) framework in [30] proposed that instead of finetuning all the parameters in a well-trained (pretrained) model, one can instead partition the network into a (global) backbone model and task-specific feature transformations. Note that similar ideas have also been explored in [28, 29, 31–33]. Given a trained backbone convolutional neural network, we can transform the convolutional feature maps $F$ for each layer into task-specific feature maps $W$ by implementing small convolutional transformations.

In our setting, only $W$ is learned for task $T_t$, as a means to reduce the parameter count, and thus the memory footprint, required for new tasks. Specifically, the feature transformation involves two types of convolutional kernels, namely, $\omega^c \in \mathbb{R}^{a \times 3 \times 3}$ for capturing spatial features within groups of channels and $\omega^s \in \mathbb{R}^{1 \times 1 \times b}$ for capturing features across channels at every location in $F$, where $a$ and $b$ are hyperparameters controlling the size of each feature map groups.
we can think of data instances where the feature maps \( F \) of groups into which \( M \) have spatial dimensions \( M \times N \times K \) and \( W \in \mathbb{R}^{M \times N \times K} \) have number of groups into which \( F \) is split for each feature map, \( \gamma \in \{0, 1\} \) indicates whether the point-wise convolutions \( \omega^d \) are employed, and \( W^s_{a:(i+a-1)} \) and \( W^{d}_{b:(i+b-1)} \) are slices of the transformed feature map. In practice, we set \( a \ll K \) and \( b \ll K \) so that the amount of trainable parameters per task is substantially reduced. For instance, using a ResNet18 backbone, \( a = 8 \) and \( b = 16 \) results in 449k parameters per new tasks, which is 3.9\% of the size of the backbone.

As empirically demonstrated in [30], EFT preserves the remarkable representation learning power of ResNet models while significantly reducing the number of trainable parameters per task.

### 3.3 Task Continual Learning: Mixture Model Perspective

One of the key components when trying to identify similar tasks is to decide whether \( \{x_i^\delta\}_{i=1}^{n_\delta} \) and \( \{x_i^\nu\}_{i=1}^{n_\nu} \) originate from the same distribution \( P \). However, though conceptually simple, it is extremely challenging in practice, particularly when predictors are complex instances such as images. Intuitively, for a sequence of tasks \( T = \{T_1, \ldots, T_{t-1}\} \), with corresponding data \( D_1, \ldots, D_{t-1} \), \( D_t = \{x_i^t, y_i^t\}_{i=1}^{n_t} \) consisting of \( n_t \) instances, where \( x_i^t \) is an image and \( y_i^t \) is its corresponding label, we can think of data instances \( x \) from the collection of all unique tasks as a mixture model defined as

\[
p(x) = \pi_\delta p(x|\phi_\delta) + \sum_{j=1}^{t-1} \pi_j p(x|\phi_j),
\]

from which we can see that \( \pi_j \) is the probability that \( x \) belongs to task \( T_j \) and \( p(x|\phi_j) \) is the likelihood of \( x \) under the distribution for task \( T_j \) parameterized by \( \phi_j \). Further, \( \pi_\delta \) and \( \phi_\delta \) denote the hypothetical probability and parameters for a new unseen task \( \delta \), i.e., distinct from \( \{p(x|\phi_j)\}_{j=1}^{t-1} \). The formulation in [3] which is reminiscent of a Dirichlet Process Mixture Model (DPMM) [34, 35], can in principle be used to estimate a posteriori that \( p(D_t \in \mathcal{T}_\delta) \) by evaluating [3], which assumes that parameters \( \{\pi_\delta, \pi_1, \ldots, \pi_{t-1}\} \) and \( \{\phi_\delta, \phi_1, \ldots, \phi_{t-1}\} \) are readily available. Unfortunately, though for the collection of existing tasks \( T \) we can effectively estimate \( \pi_j \) and \( p(x|\phi_j) \), using generative models (e.g., variational autoencoders), the parameters for a new task \( \phi_\delta \) and \( p(x|\phi_\delta) \) are much more difficult to estimate because \( i \) if we naively build a generative model for the new dataset \( D_t \) to obtain \( \phi_\delta \) and then evaluate [3]. It is almost guaranteed that \( D_t \) corresponding to \( T_\delta \) will be more likely under \( p(x|\phi_\delta) \) than any other existing task distribution \( \{p(x|\phi_j)\}_{j=1}^{t-1} \), and alternatively, \( ii \) if we set \( p(x|\phi_\delta) \) to some prior distribution, e.g. a pretrained generative model, it will be most definitely never selected, especially in scenarios with complex predictors such as images. In fact, in early stages of development we empirically verified this being the case using both pretrained generative models admitting (marginal) likelihood estimation [36, 37] and anomaly detection models based on density estimators [38, 39].

In Section 4, we will show how to leverage [3] to identify new tasks in the context of TCL without using data from \( T_t \) for learning nor specifying a prior distribution for \( p(x|\phi_\delta) \).

### 3.4 Estimating the Association between Predictors and Labels

The mixture model perspective for TCL in [3] offers a compelling way to compare the distribution of predictors for different tasks, however, it does not provide any insights about the strength of the association between predictors and labels for dataset \( D_t \) corresponding to task \( T_t \). Further, [40] showed that overparameterized neural network classifiers can attain zero training error, regardless of the strength of association between predictors and labels, and in an extreme case, even for randomly labeled data. However, it is clear that a model trained with random labels will not generalize.

So motivated, [41] first studied the properties of suitably labeled data that control generalization ability, and proposed a generalization bound on the test error (empirical risk) for arbitrary overparameterized
Figure 1: SDR Framework: We start with a pretrained feature extractor backbone $E_0(\cdot)$ and models for three dissimilar tasks kept in a knowledge repository, so the following tasks have some tasks to compare to at first. Autoencoding pairs $G(\cdot)$ and $E(\cdot)$ are used for building the distributional consistency estimator and analyzing predictor-label association in the similar task identification module. The similar task identification module uses (6) and (8) in the middle block with training dataset $D_t$ for each new task $T_t$ in the CL sequences. If there is a similar task in the knowledge repository, then simply map the new task to the identified previous task, reuse the feature extractor, $E(\cdot)$, and only learn a new classification head $f_t(\cdot)$ for the current task; alternatively, we learn a new encoder, decoder and classification head. At the end of the learning for each task, the knowledge repository is updated with the newly trained parameters. $C$ denotes the number of unique tasks in the repository, which is smaller than the total number of tasks seen so far.

two-layer neural network classifiers with rectified linear units (ReLU). Importantly, unlike previous works on generalization bounds for neural networks \cite{42,43,44}, their bound can be effectively calculated without training the network or making assumptions about its size (number of hidden units). Their complexity measure, which is shown in (4), is set to directly quantify the strength of the association between data and labels without learning. More precisely, for dataset $D = \{x_i, y_i\}_{i=1}^n$ of size $n$, the generalization bound in (4) is an upper bound on the (test) error conditioned on $D$,

$$S(D) = \sqrt{\frac{2y^T H^{-1} y}{n}},$$

(4)

where $y = (y_1, \cdots, y_n)^T$ and matrix $H \in \mathbb{R}^{n \times n}$, which can be seen as a Gram matrix for a ReLU activation function is defined as

$$H_{ik} = E_{w \sim \mathcal{N}(0, I)} \left[ x_i^T x_k \mathbb{I} \{w^T x_i \geq 0, w^T x_k \geq 0\} \right] = \frac{x_i^T x_k (\pi - \arccos \langle x_i^T x_k \rangle)}{2\pi}, \forall i, k \in 1, \ldots, n,$$

(5)

where $H_{ik}$ is the $(i, k)$-th entry of $H$, $\mathcal{N}(0, I)$ denotes the standard Gaussian distribution, $w$ is a weight vector in the first layer of the two-layer neural network, and $\mathbb{I}\{\cdot\}$ the an indicator function. Empirically, \cite{41} showed that the complexity measure in (4) can distinguish between strong and weak associations between predictors and labels. Effectively, weak associations tend to be consistent with randomly labeled data, thus unlikely to generalize.

In Section 4 we will leverage (4) as a metric for similar task detection, which we will recast a measure to quantify the association between the labels for a given task and the features from encoders learned from previously seen tasks.

4 Detection and Repurposing of Similar Tasks

We now propose the Similar task Detection and Repurposing (SDR) framework for task continual learning, which though tailored to image classification tasks, can be in principle reused for other modalities, e.g., text and structured data. Specifically, the framework consists of the following two components: (i) task-specific encoders $\{E_j(\cdot)\}_{j=1}^{T-1}$ for $T = \{T_j\}_{j=1}^{T-1}$; and (ii) a mechanism for
similar task identification structured as two separate but complementary components, namely, a measure of distributional similarity between the current task $T_t$ and each of all previous tasks $\mathcal{T}$, and a predictor-to-label association similarity, each of which leverage the mixture model perspective and the complexity measure introduced in Sections 3.3 and 3.4 respectively. For the former we use task-specific generative models $\{G_j(\cdot)\}_{j=1}^{t-1}$ specified as variational autoencoders that admit (marginal) likelihood estimation via the evidence lower bound (ELBO) \cite{25,40}, and for the latter we use the measure $S(\cdot)$ in \cite{41}. The complete framework described below is briefly illustrated in Figure~\ref{fig:framework} and presented as Algorithm~\ref{algo:framework} in the Supplementary Material (SM). In a nutshell, similar task identification results in one of two outcomes, namely, a previously seen task is identified as similar to $T_t$, in which case, we will use their corresponding encoder as is, but will learn a task-specific classification head. Alternatively, if $T_t$ is deemed as a new unseen task, we will learn both a task-specific encoder, decoder (generator), and classification head. The classification head for $T_t$ is defined as $p(y|x) = f_t(E_t(x))$ and specified as a fully connected network.

4.1 Task-specific Encoder

Following the specifications for the efficient feature transformations in \cite{30}, we first specify a pretrained backbone encoder $E_0(\cdot)$, then for each new unseen task, we adapt $E_0(\cdot)$ into $E_t(\cdot)$ using an EFT module that is learned together (end-to-end) with the task-specific classification head $p(y|x) = f_t(E_t(x))$ in a supervised fashion using $D_t$, while keeping $E_0(\cdot)$ fixed.

4.2 Similar Task Identification

As previously briefly described, the procedure to identify similar tasks for a new task $T_t$ among existing $\mathcal{T} = \{T_j\}_{j=1}^{t-1}$, amounts to estimate the distributional similarity of the predictors and the strength of the association between labels and predictors.

Provided a sequence of tasks $\mathcal{T}$ with corresponding encoders and generators $\{E_j(\cdot)\}_{j=1}^{t-1}$ and $\{G_j(\cdot)\}_{j=1}^{t-1}$, respectively, we can use (3) to estimate the likelihood that predictors in $D_t$ are consistent with the generator for a previously seen task. Specifically,

$$p(\mathcal{P}_t = \mathcal{P}_j) = E_{x_i \in D_t}[p(u_i = j|x_i)] = \frac{1}{n} \sum_{i=1}^{n} \sum_{j'} \pi_j p(x_i|\phi_j) / \sum_{j'} \pi_{j'} p(x_i|\phi_{j'}) , \quad (6)$$

where in a slight abuse of notation we let $p(\mathcal{P}_t = \mathcal{P}_j)$ indicate the probability that $\mathcal{P}_t$ and $\mathcal{P}_j$ have the same predictor distributions. In practice, we use the expected lower bound (ELBO) of a variational autoencoder pair $E_m(\cdot)$ and $G_m(\cdot)$ to approximate the (marginal) likelihood $p(x_i|\phi_j)$ parameterized by $\phi_j$, which encapsulates the parameters of both $E_j(\cdot)$ and $G_j(\cdot)$, i.e., encoder and generator, respectively. However, $p(\mathcal{P}_t = \mathcal{P}_j)$ though useful for previously seen tasks $\mathcal{T}$, does not help to identify new unseen tasks that will be consistent with a hypothetical distribution $\mathcal{P}_t$ specified as $p(x_i|\phi_0)$ and parameterized by $\phi_0$. Intuitively, given a new unseen task $T_t$, $p(\mathcal{P}_t = \mathcal{P}_j)$ is likely to be a draw from a uniform distribution in $t-1$ dimensions, so when $p(\mathcal{P}_t = \mathcal{P}_j) \approx 1/(t-1)$, for $j = 1, \ldots, t-1$, the predictors for the new task are likely from a new unseen task. Alternatively, when $p(\mathcal{P}_t = \mathcal{P}_j) \rightarrow 1$, for some task $T_j$, the predictors in $D_j$ are likely to be consistent in distribution to $\mathcal{P}_j$, i.e., to predictors from dataset $D_j$.

Importantly, the distributional consistency estimator in (6) only estimates the probability that predictors in $D_t$ are consistent in distribution to that of $D_j$. However, we still need to estimate the likelihood that encoder $E_j(\cdot)$ built from $D_j$ is strongly associated with the labels of interest in $D_t$ without building a classification model for $D_t$ using $E_j(\cdot)$. For this purpose, we leverage the complexity measure in (4), which we write below in terms of the task-specific encoders.

Following the construction of $\mathbf{H}$ as defined in (5), we start by redefining it in terms of the task-specific encoders. Specifically, given encoders $\{E_j(\cdot)\}_{j=1}^{t-1}$ and dataset $D_t = \{x_i^t, y_i^t\}_{i=1}^{n_t}$ for task $T_t$, we extract features for the predictors in $D_t$ using all available encoders, i.e., $\{E_j(x_i^t)\}_{i=1}^{n_t}$ for task $T_j$. For convenience, let $e^{j,t}_i = E_j(x_i^t)$. We can rewrite the Gram matrix $\mathbf{H}^{j,t}$ corresponding to encoder $j$ and dataset $D_t$ in (5) as

$$H_{ik}^{j,t} = \frac{(e^{j,t}_i)^\top e^{j,t}_k (\pi - \arccos((e^{j,t}_i)^\top e^{j,t}_k))}{2\pi} , \quad (7)$$

which is a multi-class extension of the complexity measure for binary classification tasks introduced in \cite{41}. Note that the Gram matrix $\mathbf{H}^{j,t} \in \mathbb{R}^{n_t \times n_t}$ has the same formulation as for when $y^t = \ldots$
We evaluate our method using five performance characteristics. Due to the nature of supervised learning tasks, we measure the overall average accuracy (Acc) after learning all the tasks in the sequence. For each task, we report the test-set accuracy. The highlight of our work is the similar task detection module. Therefore, it is crucial to show how well the framework can identify correctly whether a task is new or similar to a previously learned task. We analyze the ratio of the number of correct identification (short for correct) to $N$ new tasks. As for incorrect predictions (identification mistakes), we also evaluate the percentage of times that the system misses (does not recognize) the task.

\[ y_i \in \{0, 1\}^{c \times n}, \text{ where } c \text{ is the number of classes. Let } A^{j,t} = y^T (H^{j,t})^{-1} y \in \mathbb{R}^{c \times c}, \text{ the similarity metric } S \text{ between tasks } T_i \text{ and } T_j \text{ via encoder } E_j(\cdot) \text{ is defined as:} \]

\[
S(j,t) = \sqrt{\frac{2\|A^{j,t} - A^{j,t}\|_2^2}{n_t^2},}
\]

where $\| \cdot \|_2$ is the Frobenius Norm, $n_t$ is the size of $D_t$. The proof for (7) and (8) can be found in the SM A.3.

In practice, we obtain feature sets \( \{e_i^{j,t}\}_{i=1}^{n_t} \) with all available encoders \( \{E_j(\cdot)\}_{j=1}^{t-1} \) for task collection \( T \), and calculate \( S(j,t) \) for a new task \( D_t \). Then we select \( T_a \) with \( a = \arg\min_j \{S(j,t)\}_{j=1}^{t-1} \). Further, for each data point \( i \) in \( D^t \), we obtain \( p(u_i = j|x_i) \), for \( j = 1, \ldots, t-1 \) using the ELBO to evaluate \( p(x_i|\phi_j) \). Then we can estimate the probability of consistency between \( T_i \) and \( T_j \) via (7) and select \( T_b \) with \( b = \arg\max_j \{p(P_i = P_j)\}_{j=1}^{t-1} \). Finally, if \( T_a = T_b \) then \( T_a \) is a similar task, otherwise, \( T_i \) is a dissimilar task. The SDR framework is illustrated in Figure 1 from which we see that for the collection of currently learned tasks, we keep a knowledge repository containing (for the tasks seen so far) \( T \) \( E_0(\cdot) \), \( \{E_j(\cdot)\}_{j=1}^{t-1} \), \( \{G_j(\cdot)\}_{j=1}^{t-1} \), \( \{f_i(\cdot)\}_{j=1}^{t-1} \), i.e., the backbone encoder, task-specific encoder-decoder pairs, and the classification heads, respectively. For a more detailed view of the SDR framework see Algorithm 1 in the SM.

### 5 Experiments

We test our method on several image benchmark datasets, including EMNIST, CIFAR10, CIFAR100. Next we describe how to construct dissimilar and similar tasks.

For CIFAR100, we split all the classes into 20 tasks \( \{T_k\}_{k=1}^{20} \) with 5 classes for each task; then we divide data for each class evenly into two splits for every task to create the final dataset \( \{T_k\}_{k=1}^{20}, k \in [1, 10] \). For instance, \( T_{1,1} \) and \( T_{1,2} \) have the exact same classes, but the images come from different splits of the same classes. In general, \( T_{k,r}, \forall r \) are similar tasks for the same \( k \).

We can create CL task sequences for CIFAR10 and EMNIST in the same way as for CIFAR100. CIFAR10 has 10 classes in total, so we can evenly split the entire dataset into 5 tasks \( \{T_k\}_{k=1}^{5} \) with 2 classes for each task, and then create the mixed sequence dataset \( \{T_{k,r}\}_{r=1}^{5}, k \in [1, 5] \). For EMNIST (balanced version), to create a 100-task sequence, we divide 47 classes into 10 tasks \( \{T_{k,r}\}_{r=1}^{10}, k \in [1, 10] \) with 5 classes for \( \{T_{k,r}\}_{r=1}^{10}, k \in [1, 9] \), and 2 classes for \( \{T_{k,r}\}_{r=1}^{10}, k = 10 \). Additional details of the datasets can be found in Tables 5 and 6 in the SM A.2.

#### 5.1 Model Architecture and Experimental Details

We start the CL experiments with a backbone \( E_0(\cdot) \) and the models for three dissimilar tasks trained in the repository. The system seeks to identify if a task is new or seen to the CL system for \( N \) new tasks, with \( N+3 \) being the total number of tasks. Different task ordering in the sequence may lead to different results for the system. To show the robustness of our method to the effects of task ordering, all the experiments are repeated with five randomly permuted task sequences.

We use a pretrained ResNet18 \( \text{[47]} \) model on ImageNet \( \text{[48]} \) as our global backbone model \( E_0(\cdot) \), and for each new task \( T_i \), we train a set of task-specific parameters for \( E_i(\cdot), G_i(\cdot) \) and \( f_i(\cdot) \). The autoencoding pair for the distributional consistency estimator is implemented with a standard variational autoencoder (VAE) \( \text{[49]} \). To further reduce memory footprint, we also apply the feature transformation approach discussed in Section 4.1 to the convolutional layers in VAE models for the experiments with CIFAR10 and CIFAR100. We start with a VAE model pretrained on ImageNet \( \text{[50]} \), a subset of ImageNet, and for each new task \( T_i \), similar to \( E_i(\cdot) \), we train a set of task-specific parameters for each \( G_i(\cdot) \). More architecture and experimental details can be found in the SM A.1.

We evaluate our method using five performance characteristics. Due to the nature of supervised learning tasks, we measure the overall average accuracy (Acc) after learning all the tasks in the sequence. For each task, we report the test-set accuracy. The highlight of our work is the similar task detection module. Therefore, it is crucial to show how well the framework can identify correctly whether a task is new or similar to a previously learned task. We analyze the ratio of the number of correct identification (short for correct) to \( N \) new tasks. As for incorrect predictions (identification mistakes), we also evaluate the percentage of times that the system misses (does not recognize) the task.

We start the CL experiments with a pretrained ResNet18 \( \text{[47]} \) model on ImageNet \( \text{[48]} \) as our global backbone model \( E_0(\cdot) \), and for each new task \( T_i \), we train a set of task-specific parameters for \( E_i(\cdot), G_i(\cdot) \) and \( f_i(\cdot) \). The autoencoding pair for the distributional consistency estimator is implemented with a standard variational autoencoder (VAE) \( \text{[49]} \). To further reduce memory footprint, we also apply the feature transformation approach discussed in Section 4.1 to the convolutional layers in VAE models for the experiments with CIFAR10 and CIFAR100. We start with a VAE model pretrained on ImageNet \( \text{[50]} \), a subset of ImageNet, and for each new task \( T_i \), similar to \( E_i(\cdot) \), we train a set of task-specific parameters for each \( G_i(\cdot) \). More architecture and experimental details can be found in the SM A.1.

We evaluate our method using five performance characteristics. Due to the nature of supervised learning tasks, we measure the overall average accuracy (Acc) after learning all the tasks in the sequence. For each task, we report the test-set accuracy. The highlight of our work is the similar task detection module. Therefore, it is crucial to show how well the framework can identify correctly whether a task is new or similar to a previously learned task. We analyze the ratio of the number of correct identification (short for correct) to \( N \) new tasks. As for incorrect predictions (identification mistakes), we also evaluate the percentage of times that the system misses (does not recognize) the task.
Table 1: Task similarity prediction performance during continual learning. Results are averages over five randomly permuted CL sequences.

| SDR (EFT for $E$&$G$) | Correct | Mistake | Incorrect |
|------------------------|---------|---------|-----------|
| CIFAR10                | 88.6%   | 2.9%    | 8.6%      |
| CIFAR100               | 89.7%   | 4.3%    | 5.9%      |
| EMNIST                 | 93.8%   | 0.2%    | 6.0%      |

| SDR (EFT for $E$&$G$) |
|-----------------------|
| CIFAR10               | 97.1%   |
| CIFAR100              | 88.1%   |

similar task previously seen in the past, and that the incorrect identification (short for incorrect) to reuse to $N$ new tasks. Further, we report the overall memory usage (in MB) by the end of training.

5.2 Baselines

The CL baseline methods to which we compare are: (a) **Single Model per task**: Training a model separately for each task without finetuning, which means it will not suffer from CF. (b) **Optimal**: The optimal situation for our method, where the system always selects the right model to reuse for tasks in the sequence or trains a new set of VAE parameters and classifier for the new task. This serves as a strong upper bound for the proposed approach. (c) **Finetune**: Sequentially training a single model on all the tasks without handling forgetting issues. (d) **New-Head**: A single model with a shared feature extractor for all the tasks and a new classification head for each individual task. (e) **Online EWC [5]**: Training with a regularization term added to the loss function for a single model, so that the changes to important parameters are penalized during training for later tasks.

(f) **Experience Replay [7]**: Finetuning with the subset of exemplars (10 for each class) saved for each task for rehearsal purpose. (g) **Deep Generative Replay (DGR) [51]**: Training a generative model to replay the images. Following [51], the replayed images were labeled with the most likely category predicted by a copy of the main model stored after training on the previous task, i.e., hard targets. (h) **HAT [22]**: Learning an attention mask over the parameters in backbone to prevent forgetting. For fair comparison, we implement HAT with a wider version of the original AlexNet model used in [22]. (i) **CAT [19]**: This method considers a similar CL setting as ours. Additional baseline architecture details can be found in the SM A.1.

5.3 Results

We first show the performance of task similarity prediction module in Table 1. As a key component in our system, it demonstrates high prediction accuracy across all benchmark datasets. The performance remains competitive for CIFAR10 and CIFAR100 even after we further reduce the memory usage by applying EFT feature transformation to the generators $G(\cdot)$. We also note that, according to our experimental results, the ImageNet-pretrained VAE is not suitable as a backbone for EMNIST data, thus we do not report the corresponding results.

In Figure 2, we also present two task similarity prediction examples from the same permuted sequence of CIFAR100. The left figure shows the case where the system successfully identifies $T_3$ as a new task. Moreover, the figure on the right shows the case where the system successfully identifies $T_3$ as a similar task from the past. Task $T_3$ images are the most similar to task $T_3$ based on $p(P_t = P_j)$. Task $T_3$ consists of images in superclass “fish” from CIFAR10 while task $T_3$ includes images from superclass “aquatic mammals”. Both tasks have a substantial amount of images with a blue ocean background, and dolphins and whales look like sharks. Regardless, our similar task identification module realizes that features extracted with task $T_3$ are not the most consistent with the labels given the current learning knowledge. Concerning task $T_{19}$, from all 22 tasks kept in the knowledge repository, the system recognizes that the images for the current task are the most similar to $T_{19}$. Hence, the features extracted with the model for task $T_{19}$ can describe the labels the best.

We also compare SDR with several CL baselines in Table 2 in terms of continual learning performance and memory usage. Though the naive Single Model per task achieves highest average accuracy for
Figure 2: Examples from the same permuted CIFAR100 task sequence. The left y-axis shows $p(P_t = P_j)$ of the current task with previous tasks. The right y-axis shows the value of $S(j, t)$ for each previous task $j$ with current task $t$. The x-axis represents all the previously identified dissimilar tasks by the system. On the right, we show a list of the 20 superclass labels, the indexes of which correspond to $k$ in task $T_{k,r}$.

Table 2: Comparing SDR with continual learning baselines. Results are averages over five randomly permuted CL sequences.

| Method                  | CIFAR10 (10 tasks) | CIFAR100 (40 tasks) | EMNIST (100 tasks) |
|-------------------------|--------------------|---------------------|--------------------|
|                         | Acc(%)            | Mem(MB)            | Acc(%)            | Mem(MB)            | Acc(%)            | Mem(MB)            |
| Optimal (EFT for $E$)   | 88.45             | 76.2                | 68.30             | 169.2              | 97.41             | 107.0              |
| Optimal (EFT for $E$ and $G$) | 88.45 | 64.7                | 68.30             | 114.0              | -                 | -                  |
| Single Model per task   | 92.28             | 453.0               | 72.97             | 1812.0             | 97.39             | 4530.0             |
| Finetune                | 50.92             | 45.3                | 21.94             | 45.3               | 32.84             | 45.3               |
| New-Head                | 56.10             | 45.3                | 23.22             | 45.7               | 36.93             | 46.3               |
| ER                      | 78.22             | 55.7                | 24.83             | 61.2               | 43.36             | 71.0               |
| DGR                     | 63.70             | 95.0                | 21.79             | 95.0               | 25.16             | 95.0               |
| Online EWC              | 79.54             | 140.4               | 43.31             | 140.4              | 89.89             | 140.4              |
| CAT                     | 79.52             | 210.8               | 50.20             | 220.8              | 93.27             | 200.0              |
| HAT                     | 86.63             | 104.7               | 37.09             | 108.7              | 93.76             | 114.7              |
| SDR (EFT for $E$)       | 87.25             | 77.4                | **67.17**         | 176.8              | **96.26**         | 109.0              |
| SDR (EFT for $E$ and $G$)| **88.64** | 65.4                | 66.94             | 119.2              | -                 | -                  |

In this study, we explored a practical TCL context where tasks are not always distinct from each other. We propose a mechanism that allows for the identification of previously seen tasks to avoid repetitive training; as a means to address memory issues with parameter expansion methods for TCL. The proposed mechanism is analyzed across several image benchmarks in terms of identifying reusable modules (encoders and generators) from previous experience. Our experimental results showed promising performance even without any regularization or special modulation to the predictive models. For future work, we intend to explore the possibility of integrating our task similarity detection module in even more realistic Cl settings, where there is class overlap between tasks.
References

[1] Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258, 2017.

[2] James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114:3521 – 3526, 2017.

[3] Hongjoon Ahn, Sungmin Cha, Donggyu Lee, and Taesup Moon. Uncertainty-based continual learning with adaptive regularization. In *NeurIPS*, volume 32, 2019.

[4] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. In *ICLR*, 2018.

[5] Jonathan Schwarz, Wojciech M. Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *ICML*, volume abs/1805.06370, 2018.

[6] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, G. Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, pages 5533–5542, 2017.

[7] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Greg Wayne. Experience replay for continual learning. In *NeurIPS*, 2019.

[8] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *AAAI*, 2018.

[9] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *NIPS*, 2017.

[10] Gido M. van de Ven and Andreas Savas Tolias. Generative replay with feedback connections as a general strategy for continual learning. *ArXiv*, abs/1809.10635, 2018.

[11] Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *ICML*, 2019.

[12] Soochan Lee, Junsoo Ha, Dongsu Zhang, and Gunhee Kim. A neural dirichlet process mixture model for task-free continual learning. In *ICLR*, 2019.

[13] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Kwang. Lifelong learning with dynamically expandable networks. In *ICLR*, 2018.

[14] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *ArXiv*, abs/1606.04671, 2016.

[15] Tom Véniat, Ludovic Denoyer, and Marc’Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors. In *ICLR*, volume abs/2012.12631, 2021.

[16] Oleksiy Ostapenko, Pau Rodríguez López, Massimo Caccia, and Laurent Charlin. Continual learning via local module composition. In *NeurIPS*, volume abs/2111.07736, 2021.

[17] Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines, 2018.

[18] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Timme Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

[19] Zixuan Ke, Bing Liu, and Xingchang Huang. Continual learning of a mixed sequence of similar and dissimilar tasks. In *NeurIPS*, volume 33, pages 18493–18504, 2020.
[20] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009.
[21] Gregory Cohen, Saeed Afshar, Jonathan C. Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters. In CVPR, volume abs/1702.05373, 2017.
[22] Joan Serrà, Didac Surís, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In ICML, volume abs/1801.01423, 2018.
[23] Prakhar Kaushik, Alex Gain, Adam Kortylewski, and Alan Loddon Yuille. Understanding catastrophic forgetting and remembering in continual learning with optimal relevance mapping. ArXiv, abs/2102.11343, 2021.
[24] Gido M. van de Ven and Andreas Savas Tolias. Three scenarios for continual learning. ArXiv, abs/1904.07734, 2019.
[25] Ameya Prabhu, Philip Torr, and Puneet Dokania. Gdumb: A simple approach that questions our progress in continual learning. In ECCV, August 2020.
[26] Ahmet Iscen, Jeffrey O. Zhang, Svetlana Lazebnik, and Cordelia Schmid. Memory-efficient incremental learning through feature adaptation. ECCV, abs/2004.00713, 2020.
[27] Sakshi Varshney, Vinay Kumar Verma, Srijith P K, Lawrence Carin, and Piyush Rai. Cam-gan: Continual adaptation modules for generative adversarial networks. In NeurIPS, 2021.
[28] Yulai Cong, Miaoyun Zhao, Jianqiao Li, Sijia Wang, and Lawrence Carin. Gan memory with no forgetting. In NeurIPS, volume 33, pages 16481–16494, 2020.
[29] Zichen Miao, Ze Wang, Wei Chen, and Qiang Qiu. Continual learning with filter atom swapping. In International Conference on Learning Representations, 2022.
[30] Vinay Kumar Verma, Kevin J Liang, Nikhil Mehta, Piyush Rai, and Lawrence Carin. Efficient feature transformations for discriminative and generative continual learning. In CVPR, pages 13860–13870, 2021.
[31] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. ArXiv, abs/1704.04861, 2017.
[32] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In AAAI, 2018.
[33] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In CVPR, pages 7132–7141, 2018.
[34] Samuel J. Gershman and David M. Blei. A tutorial on bayesian nonparametric models. Journal of Mathematical Psychology, 56:1–12, 2011.
[35] David M Blei and Michael I Jordan. Variational inference for dirichlet process mixtures. Bayesian analysis, 1(1):121–143, 2006.
[36] Irina Higgins, Loïc Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew M. Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In ICLR, 2017.
[37] Christopher P. Burgess, Irina Higgins, Arka Pal, Loïc Matthey, Nicholas Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-VAE. In NIPS, 2017.
[38] Eric Pauwels and Onkar Ambekar. One class classification for anomaly detection: Support vector data description revisited. In ICDM, pages 25–39, 08 2011.
[39] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In ICML, pages 4393–4402. PMLR, 2018.
[40] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. In CACM, volume 64, pages 107–115. ACM New York, NY, USA, 2021.

[41] Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In ICML. PMLR, 2019.

[42] Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In UAI, volume abs/1703.11008, 2017.

[43] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P. Adams, and Peter Orbanz. Non-vacuous generalization bounds at the imagenet scale: a pac-bayesian compression approach. In ICLR, 2019.

[44] Xingguo Li, Junwei Lu, Zhaoran Wang, Jarvis D. Haupt, and Tuo Zhao. On tighter generalization bound for deep neural networks: Cnns, resnets, and beyond. In ICLR, volume abs/1806.05159, 2019.

[45] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In ICLR, volume abs/1312.6114, 2014.

[46] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. Journal of the American Statistical Association, 112:859 – 877, 2016.

[47] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In CVPR, pages 770–778, 2016.

[48] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In CVPR, pages 248–255. Ieee, 2009.

[49] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. ArXiv, abs/2003.05991, 2020.

[50] Jeremy Howard. Imagenette.

[51] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In NIPS, 2017.

[52] Simon Shaolei Du, Xiyu Zhai, Barnabás Póczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. ICLR, abs/1810.02054, 2019.
Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] The continual learning scenario we discuss in the paper is just one practical case of real-world settings. In Section 6, we claimed that we are going to explore even more practical settings where there is class overlap between tasks.
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] The assumptions of theoretical results can be found in Section 3.
   (b) Did you include complete proofs of all theoretical results? [Yes] It is shown in the SM A.3.

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code will be released upon acceptance.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Model architecture and experimental details, including data description, hyperparameter selection are provided in the SM A.2 and A.1.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] The corresponding performance is provided in the SM.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] We have not, however, these details will be included in the final version.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] The code reference is not mentioned in the main paper, but we will provide the license of the assets in our code base acknowledgement section.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No] The CL tasks sequences will be released with the source code.
   (d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [No] The data we are using in the paper is open-source.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] There is no personally identifiable information or offensive content in the data we are using.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]
A  Appendix

A.1  Experimental Details

In all the experiments except for methods CAT and HAT, we use a pretrained ResNet18 as the backbone of the classification models. To adapt the same pretrained backbone model for EMNIST, which contains images of one channel, we first transform the grey-scale images into three-channel images. For a fair comparison, we adopt the efficient feature transformation technique in Optimal, Finetune, New-Head, and SDR. Serving as a benchmark, Single Model per task is trained with a full ResNet18 for each task. They all use the pretrained ResNet18 on ImageNet as a backbone. For CAT, we show the results with a 2 fully connected layer network used in the original paper. Following the setting in CAT paper [19], the embeddings for the hidden and final layer in the knowledge base have a dimension of 2000; the task ID embeddings also have 2000 dimensions. For HAT, we switched to a wide version of AlexNet model as backbone. Specifically, each layer in the wide version has twice the number of nodes compared to each layer in the original version [22]. For DGR, we follow the setting in [10] for the generative model, which is a symmetric VAE with 3 fully connected layers (fc1000 - fc1000 - fc100). As for the VAE models in SDR, an architecture of 4 convolutional layers and 1 linear layer is used for both encoder and decoder.

All the classification models (when feature extractors are newly trained) are trained with a batch size of 128 for 100 epochs, with a starting learning rate of 0.01 reduced with a factor of 0.1 at epoch 50. The only data augmentation/ transformation used for the datasets (except for EMNIST, which is converted from one-channel to three-channels) is normalization. When the feature extractor is reused, the classification head only needs to be trained for 2 epochs with a learning rate of 0.001. As for VAE models trained in SDR, they are trained with a batch size of 64 and a learning rate of 0.0001, for 2000 epochs with early stopping criteria.

Here we provide a detailed view of the SDR framework in 1.

A.2  Dataset Details

In Table 3, we provide the number of tasks in the sequences created with each dataset, the number of classes for each task, and the size of each data split.

| Dataset     | # tasks | # classes/tsk | training/tsk | validation/tsk | testing/tsk |
|-------------|---------|----------------|--------------|----------------|-------------|
| CIFAR10     | 10      | 2              | 4500         | 500            | 1000        |
| CIFAR100    | 40      | 5              | 1125         | 125            | 250         |
| EMNIST      | 100     | 5 or 2         | 1080 or 432  | 120 or 48      | 200 or 80   |

A.3  Proof

In Section A.3.1 to A.3.3 we provide theoretical proofs for our similarity metric, while in Section A.3.4 we show empirical proofs.

A.3.1  Setting

Given a two-layer ReLU activated neural network with $m$ neurons in the hidden layer and $c$ neurons in the output layer:

\[ f_{W,A}(x) = \frac{1}{\sqrt{m}} \sum_{r=1}^{m} a_r \sigma(w_r^T x), \]  

(9)

where $x \in \mathbb{R}^d$ is an input, $W = (w_1, \cdots, w_m) \in \mathbb{R}^{d \times m}$ are the weight vectors in the first layer and $A = (a_1, \cdots, a_m)^T \in \mathbb{R}^{c \times m}$ are weight vectors in the second layer, with $c$ being the number of classes.
Algorithm 1 SDR Framework

Data: $D_t = \{x_t^i, y_t^i\}^{n_t}_{i=1}, t \in [1, N]$; (Only $D_t$ available at time $t$)
Result: $\{E_m\}^C_{m=1} = \{G_m\}^C_{m=1}$, with $C$ being the number of unique tasks recognized by the system

Starting with: $E_0$, $\{E_m\} \leftarrow \{E_m\}^3_{m=1}$; $G_0$ (might or might not be available), $\{G_m\} \leftarrow \{G_m\}^3_{m=1}$

for $t \leftarrow 1$ to $N$ do
  for $j \leftarrow 1$ to $t - 1$ do
    for $i \leftarrow 1$ to $n^j$ do
      $p(u_i = j|x_i); \quad /*$ Distributional Consistency Estimator */
    end
    $p(\mathcal{P}_t = \mathcal{P}_j) = \mathbb{E}_{x_i \in D_t}[p(u_i = j|x_i)]$
  end
  $T_a = \arg\max_j \{p(\mathcal{P}_t = \mathcal{P}_j)\}^{t-1}_{j=1}$
  for $j \leftarrow t - 1$ to 1 do
    $\mathbf{H}^{t,j}_{ik} = \frac{\partial^2}{\partial \theta^{i,j} \cdot \partial \theta^{k,t}} (\pi \cdot \arccos(\mathbf{e}^{i,j}^T \mathbf{e}^{k,t}))$
    $\mathbf{A}^{j,t} = \mathbf{y}^T (\mathbf{H}^{t,j})^{-1} \mathbf{y}$
    $S(j, t) \leftarrow \sqrt{\frac{2||\mathbf{A}^{j,t}||^2}{n_t}}; \quad /*$ Predictor-label Association Analysis */
  end
  $T_b = \arg\min_{j \in 1} \{S(j, t)\}^{t-1}_{j=1}$
  if $T_a == T_b$ then
    reuse feature encoder and VAE model from task $T_a$ for task $T_t$
  else train new models for task $T_t$
    repeat
      $\mathbf{B} \leftarrow$ random sampled mini-batch from $D_t$
      $g = \nabla_{\theta_{E_t}} \mathcal{L}(\phi_{E_t}, \mathbf{B})$
      $\theta_{E_t} \leftarrow \text{AdamOptimizer}(\theta_{E_t}, g)$
      until $\theta_{E_t}$ converges;
    repeat
      $\mathbf{B} \leftarrow$ random sampled mini-batch from $D_t$
      $g = \nabla_{\psi_{G_t}, \phi_{G_t}} \mathcal{L}(\phi_{G_t}, \psi_{G_t}, \mathbf{B})$
      $\psi_{G_t}, \phi_{G_t} \leftarrow \text{AdamOptimizer}(\psi_{G_t}, \phi_{G_t}, g)$
      until $\psi_{G_t}, \phi_{G_t}$ converges;
    end
end

We train the neural network by randomly initialized gradient descent (GD) on the quadratic loss over data. In particular, we first initialize the parameters randomly[1]

$$w_r(0) \sim \mathcal{N}(0, \kappa^2 I), \quad \sum_{z=1}^c a_{z,r} \sim \text{unif}(\{-1, 1\}), \quad \forall r \in [m],$$

where $0 < \kappa \leq 1$ controls the magnitude of initialization.

We want to minimize quadratic loss of

$$\mathcal{L}(\mathbf{W}) = \frac{1}{2N} \sum_{i=1}^N \sum_{z=1}^c (y_{iz} - f_{W,a_z}(x_i))^2,$$

through GD by fixing the second layer $\mathbf{A}$ and only optimizing the first layer $\mathbf{W}$

$$\frac{\partial \mathcal{L}(\mathbf{W})}{\partial w_r} = \frac{1}{2N} \sum_{z=1}^c a_{z,r} \sum_{i=1}^N \mathbb{I}(f_{W,a_z}(x_i) - y_{iz}) \mathbb{I}(w_r^T x_i \geq 0) x_i, \quad \forall r \in [m].$$

[1]For each $r \in [1, m]$, randomly pick sum $s_r$ of the $r$-th row $a_r$ from unif($\{-1, 1\}$). For the first $c-1$ neurons, we randomly sample $a_{c,r} \sim \text{unif}(\{-1, 1\})$. For all $r = 1, \ldots, c-1$, the last neuron is set to satisfy the column sum, i.e., $a_{c,r} = s_r - \sum_{z=1}^{c-1} a_{z,r}$.
Table 4: Details for Classes in Each Task

| Dataset  | k | Classes                                                                                     |
|----------|---|----------------------------------------------------------------------------------------------|
| CIFAR10  |   | [‘airplane’, ‘bird’]                                                                         |
|          | 2 | [‘automobile’, ‘truck’]                                                                      |
|          | 3 | [‘cat’, ‘dog’]                                                                               |
|          | 4 | [‘deer’, ‘horse’]                                                                            |
|          | 5 | [‘frog’, ‘ship’]                                                                             |
| CIFAR100 |   | [‘aquarium fish’, ‘flatfish’, ‘ray’, ‘shark’, ‘trout’]                                        |
|          | 2 | [‘orchid’, ‘poppy’, ‘rose’, ‘sunflower’, ‘tulip’]                                             |
|          | 3 | [‘beaver’, ‘dolphin’, ‘otter’, ‘seal’, ‘whale’]                                              |
|          | 4 | [‘bottle’, ‘bowl’, ‘can’, ‘cup’, ‘plate’]                                                     |
|          | 5 | [‘bear’, ‘leopard’, ‘lion’, ‘tiger’, ‘wolf’]                                                  |
|          | 6 | [‘apple’, ‘mushroom’, ‘orange’, ‘pear’, ‘sweet pepper’]                                       |
|          | 7 | [‘hamster’, ‘mouse’, ‘rabbit’, ‘shrew’, ‘squirrel’]                                           |
|          | 8 | [‘baby’, ‘boy’, ‘girl’, ‘man’, ‘woman’]                                                       |
|          | 9 | [‘maple tree’, ‘oak tree’, ‘palm tree’, ‘pine tree’, ‘willow tree’]                           |
|          | 10| [‘bicycle’, ‘bus’, ‘motorcycle’, ‘pickup truck’, ‘train’]                                     |
|          | 11| [‘bed’, ‘chair’, ‘couch’, ‘table’, ‘wardrobe’]                                                |
|          | 12| [‘bee’, ‘beetle’, ‘butterfly’, ‘caterpillar’, ‘cockroach’]                                    |
|          | 13| [‘beaver’, ‘dolphin’, ‘otter’, ‘seal’, ‘whale’]                                              |
|          | 14| [‘bottle’, ‘bowl’, ‘can’, ‘cup’, ‘plate’]                                                     |
|          | 15| [‘bear’, ‘leopard’, ‘lion’, ‘tiger’, ‘wolf’]                                                  |
|          | 16| [‘apple’, ‘mushroom’, ‘orange’, ‘pear’, ‘sweet pepper’]                                       |
|          | 17| [‘hamster’, ‘mouse’, ‘rabbit’, ‘shrew’, ‘squirrel’]                                           |
|          | 18| [‘baby’, ‘boy’, ‘girl’, ‘man’, ‘woman’]                                                       |
|          | 19| [‘maple tree’, ‘oak tree’, ‘palm tree’, ‘pine tree’, ‘willow tree’]                           |
|          | 20| [‘bicycle’, ‘bus’, ‘motorcycle’, ‘pickup truck’, ‘train’]                                     |

Define $y_{iz} = f_{W,a_i}(x_i)$, i.e., the network’s prediction on the $i$-th input that belongs to class $z$, and (12) can be rewritten as:

$$
\frac{\partial L(W)}{\partial w_r} = \frac{1}{2N} \sum_{z=1}^{c} a_{z,r} \sum_{i=1}^{N} (y_{iz} - y_{iz}) \mathbb{I}_{r,i} x_i, \quad \forall r \in [m],
$$

where $\mathbb{I}_{r,i} = \mathbb{I}\{w_r^\top x_i \geq 0\}$. We also define $Z$ as:

$$
Z = \frac{1}{\sqrt{m}} \begin{bmatrix}
\mathbb{I}_{1,1}(\sum_{z=1}^{c} a_{z,1}) x_1 & \cdots & \mathbb{I}_{1,1}(\sum_{z=1}^{c} a_{z,m}) x_n \\
\vdots & \ddots & \vdots \\
\mathbb{I}_{m,1}(\sum_{z=1}^{c} a_{z,1}) x_1 & \cdots & \mathbb{I}_{m,1}(\sum_{z=1}^{c} a_{z,m}) x_n
\end{bmatrix}
$$

With this notation we have a more compact form of the gradient (12):

$$
\text{vec}(\nabla(L(W))) = Z(y - y).
$$
\[ H = Z^\top Z^2 \] since we initialize the second layer with \( \sum_{z=1}^{c} a_{z,r} \sim \text{unif}\{\{-1, 1\}\} \)

\[ H_{ij} = \frac{x_i x_j}{m} \sum_{r=1}^{m} \|r_{r,1} r_{r,2}\left(\sum_{z=1}^{c} a_{z,r}\right)^2 \]

\[ = \frac{x_i x_j}{m} \sum_{r=1}^{m} \|r_{i,r} r_{j,r}\}, \quad \forall i, j \in [n] \quad (17) \]

and thus we prove that we can obtain the Gram matrix \( H \) the same form for multi-classification setting as in binary-classification settings, and complete the proof of (7).

### A.3.2 Rademacher Complexity and Generalization

Generalization error measures how accurate an algorithm is for predicting outcome values for previously unseen data. Specifically, given a function \( f : \mathbb{R}^d \to \mathbb{R} \) and a loss function \( l : \mathbb{R} \times \mathbb{R} \to \mathbb{R} \), generalization error can be defined as the gap between the population loss over data distribution \( P \) and the empirical loss over \( n \) samples \( D = \{(x_i, y_i)\}_{i=1}^{n} \) from \( P \):

\[ \mathcal{L}_P(f) - \mathcal{L}_D(f) = \mathbb{E}_{(x,y)\sim P} [l(f(x,y))] - \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i) \quad (18) \]

**Definition A.1** ([41]) Given \( n \) samples \( D \), the empirical Rademacher complexity of a function class \( \mathcal{F} \) (mapping from \( \mathbb{R}^d \)) is defined as:

\[ \mathcal{R}_D(\mathcal{F}) = \frac{1}{n} \mathbb{E}_\epsilon \left[ \sup_{f \in \mathcal{F}} \sum_{i=1}^{n} \epsilon_i f(x_i) \right], \]

where \( \epsilon = (\epsilon_1, \cdots, \epsilon_n)^\top \) contains i.i.d random variables drawn from the Rademacher distribution \( \text{unif}\{\{-1, 1\}\} \).

Rademacher complexity directly gives an upper bound on generalization error:

**Theorem A.1** ([41]) Suppose the loss function \( l(\cdot, \cdot) \) is bounded in \([0, c]\) and is \( \rho \)-Lipschitz in the first argument. Then with probability at least \( 1 - \delta \) over sample \( D \) of size \( n \):

\[ \sup_{f \in \mathcal{F}} \{\mathcal{L}_P(f) - \mathcal{L}_D(f)\} \leq 2\rho \mathcal{R}_D(\mathcal{F}) + 3c \sqrt{\frac{\log(2/\delta)}{2n}} \quad (19) \]

Therefore, as long as we can bound the Rademacher complexity of a certain function class that contains our learned predictor, we can obtain a generalization bound.

### A.3.3 Proof for Complexity Measure under Multi-class Scenario

In this section, we show proof to the complexity measure \( \mathcal{S} \) under multi-class classification settings. We note that most of the related proofs of complexity measure (generalization bound) under binary classification setting are provided in [41].

**Theorem A.2** ([52], [41]) Assume \( \lambda_0 = \lambda_{\min}(H^\infty) > 0 \). For \( \delta \in (0, 1) \), if \( m = \Omega \left( \frac{n^6}{\lambda_0^2 \eta^2} \right) \) and \( \eta = O \left( \frac{\lambda_0}{n^2} \right) \), then with probability at least \( 1 - \delta \) over the random initialization, we have:

- \( \mathcal{L}(W(0)) = O(n/\delta) \);
- \( \mathcal{L}(W(k + 1)) \leq \left( 1 - \frac{\eta \lambda_0}{2} \right) \mathcal{L}(W(k)), \quad \forall k \geq 0. \)

---

\( H \) corresponds to \( H^\infty \) in [41], which denotes the Gram matrix when GD converges, here we abuses the notation to be consist with the main paper.
Analysis of the Auxiliary Sequence \( \{ \tilde{W}(k) \}_{k=0}^{\infty} \) \[ \text{Now we give a proof of } \| \tilde{W}(\infty) - \tilde{W}(0) \|_F = \sqrt{\| y^\top H(0)^{-1} y \|_F} \] as an illustration for the proof of Lemma A.1 Define \( v(k) = Z(0)^\top \text{vec}(\tilde{W}(k)) \in \mathbb{R}^n \). Then from (20) we have \( v(0) = 0 \) and \( v(k+1) = v(k) - \eta H(0) (v(k) - y) \), yielding \( v(k) - y = -(I - \eta H(0))^k y \). Plugging this back to (20) we get
\[
\text{vec} \left( \tilde{W}(k+1) \right) - \text{vec} \left( \tilde{W}(k) \right) = \eta Z(0) (I - \eta H(0))^k y.
\]
Then taking a sum over step \( k = 0, 1, \ldots \) we have
\[
\text{vec} \left( \tilde{W}(\infty) \right) - \text{vec} \left( \tilde{W}(0) \right) = \sum_{k=0}^{\infty} \eta Z(0) (I - \eta H(0))^k y = Z(0) H(0)^{-1} y.
\]

The desired result thus follows\[ 4 \]
\[
\| \tilde{W}(\infty) - \tilde{W}(0) \|_F^2 = y^\top H(0)^{-1} Z(0)^\top Z(0) H(0)^{-1} y = y^\top H(0)^{-1} y.
\]
\[
\| \tilde{W}(\infty) - \tilde{W}(0) \|_F = \sqrt{\| y^\top H(0)^{-1} y \|_F}.
\]

Lemma A.1’s proof is based on the careful characterization of the trajectory of \( \{ W(k) \}_{k=0}^{\infty} \) during GD. In particular, we bound its distance to initialization as follows:

**Lemma A.1 (41)** Suppose \( m \geq \kappa^{-2} \text{poly}(n, \lambda_0^{-1}, \delta^{-1}) \) and \( \eta = O \left( \frac{\kappa^2}{m} \right) \). Then with probability at least \( 1 - \delta \) over the random initialization, we have for all \( k \geq 0 \):

- \( \| w_r(k) - w_r(0) \|_2 \leq O \left( \frac{n}{\sqrt{m \lambda_0 \sqrt{\delta}}} \right) (\forall r \in [m]), \)
- \( \| W(k) - W(0) \|_F \leq \sqrt{\| y^\top (H^{\infty})^{-1} y \|_F} + O \left( \frac{n \kappa}{\lambda_0^2} + \frac{\text{poly}(n, \lambda_0^{-1}, \delta^{-1})}{m^{1/2} \kappa^{1/2} \delta} \right). \)

The bound on the movement of each \( w_r \) was proved in [52]. The bound on \( \| W(k) - W(0) \|_F \) corresponds to the total movement of all neurons. The main idea is to couple the trajectory of \( \{ W(k) \}_{k=0}^{\infty} \) with another simpler trajectory \( \{ \tilde{W}(k) \}_{k=0}^{\infty} \) defined as:

\[
\tilde{W}(0) = 0, \quad \text{vec} \left( \tilde{W}(k+1) \right) = \text{vec} \left( \tilde{W}(k) \right) - \eta Z(0) \left( Z(0)^\top \text{vec} \left( \tilde{W}(k) \right) - y \right).
\]

**Lemma A.2 (41)** Given \( R > 0 \), with probability at least \( 1 - \delta \) over the random initialization \( (W(0), \Lambda) \), simultaneously for every \( B > 0 \), the following function class
\[
\mathcal{F}_{R,B}^{W(0),\Lambda} = \{ f_{w,\Lambda} : \| w_r - w_r(0) \|_2 \leq R (\forall r \in [m]), \| W - W(0) \|_F \leq B \}
\]
has empirical Rademacher complexity bounded as:
\[
\mathcal{R}_D \left( \mathcal{F}_{R,B}^{W(0),\Lambda} \right) \leq \frac{1}{\sqrt{2n}} \left( 1 + \left( \frac{2 \log \frac{2}{\delta}}{m} \right)^{1/4} \right) + \frac{2 R^2 \sqrt{n}}{\kappa} + R \sqrt{2 \log \frac{2}{\delta}}.
\]

\[ 3 \] Denotes the optimization step in the proof section
\[ 4 \] Note that we have \( H(0) \approx H^{\infty} \) from standard concentration. See Lemma C.3 in [41]
\[ 5 \] \( m \) is the number of parameters in the first layer, \( \eta \) is the learning rate
For the proof of Lemma A.2, the only difference compared to Lemma 5.4 in [41] is \( a_e \) in [41] becomes \( \sum_{z=1}^{n_c} a_{r,z} \) in our setting.

Finally, combining Lemma A.1 with A.2, we are able to conclude that the neural network found by GD belongs to a function class with Rademacher complexity at most \( \sqrt{\|Y(H^{\infty})^{-1}y\|_F/(2n)} \) (plus negligible errors). This gives us the generalization bound in Theorem A.3 using Rademacher complexity.

Definition A.2 A distribution \( P \) over \( \mathbb{R}^d \times \mathbb{R} \) is \((\lambda_0, \delta, n)\)-non-degenerate, if for \( n \) i.i.d. samples \( \{(x_i, y_i)\}_{i=1}^n \) from \( P \), with probability at least \( 1 - \delta \) we have \( \lambda_{\min}(H^{\infty}) \geq \lambda_0 > 0 \).

Theorem A.3 ([41]) Fix a failure probability \( \delta \in (0, 1) \). Suppose our data \( D = \{(x_i, y_i)\}_{i=1}^n \) are i.i.d. samples from a \((\lambda_0, \delta/3, n)\)-non-degenerate distribution \( P \), and \( \kappa = O(\frac{\lambda_0 \delta}{n}) \), \( n \geq \kappa^{-2} \text{poly}(n, \lambda_0^{-1}, \delta^{-1}) \). Consider any loss function \( \ell : \mathbb{R} \times \mathbb{R} \rightarrow [0, 1] \) that is 1-Lipschitz in the first argument such that \( \ell(y, y) = 0 \). Then with probability at least \( 1 - \delta \) over the random initialization and the training samples, the two-layer neural network \( f_{W(k), A} \) trained by GD for \( k \geq \Omega \left( \frac{1}{\eta \lambda_0} \log \frac{n}{\delta} \right) \) iterations has population loss \( L_P(f_{W(k), A}) = \mathbb{E}_{(x, y) \sim P} \left[ \ell(f_{W(k), A}(x), y) \right] \) bounded as:

\[
L_P(f_{W(k), A}) \leq \sqrt{\frac{2y^\top(H^{\infty})^{-1}y}{n}} + O \left( \sqrt{\frac{\log n}{n}} \right). \tag{21}
\]

A.3.4 Empirical Proof for Using Complexity Measure as our Similarity Metric

In Figures 3, 4, 5, we intent to show that datasets encoded with feature extractors trained on similar tasks obtain relatively smaller values compared to those trained on dissimilar tasks. In the heatmaps, the y-axis represents the current tasks, while the x-axis shows the previous tasks. The value of our similarity metric is color-coded for each cell in the heatmaps. The cells with smaller values are the y-axis represents the current tasks, while the x-axis shows the previous tasks. The value of our similarity metric is color-coded for each cell in the heatmaps. The cells with smaller values are

![Figure 3: Similarity Metric Results for CIFAR10](image)

Figure 3: Similarity Metric Results for CIFAR10
Figure 4: Similarity Metric Results for CIFAR100

Figure 5: Similarity Metric Results for EMNIST