A Related Works: Continued

Self Supervised Learning: SSL divides into two primary sub-areas: 1- Generative self-supervised learning, in which models predict missing data components (e.g., inpainting) [15], and 2- Discriminative self-supervised learning, where models learn representations by distinguishing between different views of identical data (e.g., contrastive learning) [15]. As the focus of this work is discriminative SSL, we will offer comprehensive literature on this aspect and utilize SSL to specifically denote discriminative SSL throughout this work.

Recent works in SSL are predominantly categorized into three families: sample contrastive, asymmetric network and dimension-contrastive [43]. Sample-contrastive techniques, exemplified by SimCLR [8], MoCO [17], SwAV [4], and FroSSL [43], treat to different views of a sample (i.e. different augmented versions) as positive samples, and any other sample as negative. Then, a contrastive loss is employed to bring positive samples closer while pushing negative samples apart. Asymmetric-network approaches include SimSiam [9], BYOL [14], and DINO [5]. These methods require distinct network architectures for input views. While one network serves as the primary network for final use, the other can adopt a different encoder structure, or stop-gradient techniques can be employed within the same architecture as the primary network [14]. Dimension-contrastive methods, such as Barlow-Twins [48], VicReg [2], W-MSE [12], CorInfoMax [36], and FroSSL [43], focus on reducing redundancy within embedding dimensions. Such approaches may eliminate the need for negative samples and the requirement for an asymmetric network structure.

Continual Learning: Continual Learning refers to Continual Supervised Learning generally. The main problem of Continual learning is catastrophic forgetting, and recent methods developed various strategies to solve that problem [11]. These methods can be broadly classified into three distinct approaches:

1. **Rehearsal-based Approaches:** These methods mitigate forgetting by replaying data from previous tasks, either stored in a limited memory [7,16,33,37] or synthesized by generative models [42]. Notably, [29] employs an experience replay technique, utilizing a stored model from a preceding task as a 'teacher' for knowledge distillation on features pre-classification.
2. **Expansion-based Approaches:** To prevent forgetting, these methods dynamically increase the network's capacity in response to new tasks [38,40,46,47]. The most crucial disadvantage of expansion-based methods is large model sizes when the new tasks appear continuously.
3. **Regularization-based Approaches:** This category encompasses methods that adjust the optimization process by introducing task-specific regularization. Such regularization aims to align the optimal parameters of new tasks with those of prior tasks, thereby minimizing forgetting [24,32,35,41]. A notable variation within this approach is the Gradient Projection Memory (GPM) technique [39], which adjusts model gradients on a per-layer basis to ensure updates are orthogonal to the gradient subspace of preceding tasks,

thus preserving prior learning. Federated Orthogonal training (FOT) [1] is another work mainly proposed for distributed learning settings that modify the global updates of new tasks so that they are orthogonal to previous tasks' activation principal subspace.

Although these methods listed above show impressive results on Continual Supervised Learning, they are not that effective in CSSL [6,13].

B Task Confusion Experiments

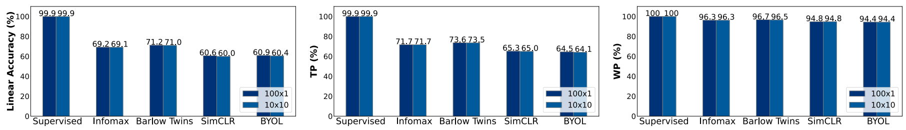
In Section 3, we stated our hypothesis, which is as follows,

The task confusion problem in Contrastive SSL methods arises primarily from the inability to train the model with different classes belonging to different tasks concurrently.

Here, we perform a data-incremental learning-based ablation study to further analyze our hypothesis.

Data-Incremental Learning: An Ablation Study

In this ablation study, we aim to analyze that the performance drop in LA and TP observed in Table 4 was because of the class split across tasks, not because of the data split across tasks. For that, we explore both self-supervised and supervised learning in another fairly simple but representative data-incremental setup, where each task has all classes data but data is split across tasks. To be more specific, our experiment setup is as follows: we follow a data-incremental setup with a sequence of tasks $\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_T$ that have same set of classes. We consider the CIFAR100 dataset and split the 50,000 data in 10 tasks with each task containing all 100 classes data and total of 5,000 samples per task. Further, we assume that tasks change after each iteration, i.e., mini-batches are sampled from different tasks at each iteration. To remove the forgetting effect, we assume that tasks can be revisited, i.e., task 2 follows task 1, task 3 follows task 2, and so on. The repeatability of tasks ensures that the SSL learner does not forget the previous knowledge. For simplicity, we refer to this experimental setup, 10x10 data-incremental learning across mini-batches, 10x10 DIL-minibatch because the data is divided into 10 tasks where each task contains 5000 samples. Further, to show how the performance of the methods changes in DIL-minibatch setting, we also compare it to the regular setting where we sample uniformly random from the whole training data. We call this regular training setting as 100x1 DIL-minibatches because there is 1 task containing all 100 classes and all the data. Here again, we report the training accuracy of the methods because we only care about the methods' capability of creating linearly separable features on the data they trained on. The training accuracy of these methods is reported in Figure 5. If data split across batches/tasks would have been the reason for the accuracy drop, we would observe an accuracy drop in these settings as well. However, in contrast to what we observed in Figure 4, we did not observe any accuracy drop



(a) LA: 10x10 versus 100x1 **(b)** TP: 10x10 versus 100x1 **(c)** WP: 10x10 versus 100x1

Fig. 5: Training LA, WP, and TP performance of contrastive SSL methods and supervised learning on the CIFAR100 Dataset for both 100x1 and 10x10 DIL-minibatch settings. Figures (a), (b), and (c) demonstrate that the 10x10 setting performs as good as the 100x1 setting in terms of LA, TP, and WP, respectively.

in average linear accuracy between 10x10 and 100x1 DIL experiment settings for both supervised as well as self-supervised learning settings. This reaffirms our hypothesis that the task confusion problem in Contrastive SSL methods arises primarily from the inability to train the model with different classes belonging to different tasks concurrently.

C Continual Learning Experiments

In this section, we provide further evaluations: an ablation study to compare different design components of CroMo-Mixup, buffer size versus accuracy performance, the model's generalization performance, and catastrophic forgetting analysis of CroMo-Mixup with the progression of time.

C.1 An Ablation Study: Different Design Components' Performance

In Table 3, we evaluate three key components of our design model, cross-task data mixup, cross-model feature mixup, and distillation (ζ) on CIFAR100-Split5 with Barlow-Twins. First, we compare the cross-task data mixup without adding any other component. For input mixup comparison, we have two choices: within-task data mixup and cross-task data mixup. We find that the cross-task data mixup enhances the performance by 8% accuracy gain as compared to the within-task setting, as shown in the first two rows of Table 3. Next, we compare output mixup, again we have two cases, the same model feature mixup that uses only the current model to obtain embeddings, and cross-model feature mixup, the proposed formulation. We note that the cross-model feature mixup enhances the performance by 1.6% accuracy gain as compared to the same-model mixup for the cross-task data mixup. This highlights the significance of the proposed CroMo-Mixup formulation. Additionally, we also notice that distillation improves the performance by another 1.5% accuracy gain with these components. Hence, cross-task mixup, cross-model mixup, and distillation components help in achieving the highest performance among other design choices of these components.

Table 3: An Ablation Study on Different Design Components of CroMo-Mixup

Input Mixup	Output Mixup	Distillation (ζ)	Accuracy(%)
within-task	same-model	0	54.16
cross-task	same-model	0	62.31
cross-task	same-model	0	62.31
cross-task	cross-model	0	63.94
cross-task	same-model	1	64.35
cross-task	cross-model	1	65.48

C.2 Buffer Size versus Accuracy Performance

To analyze the impact of buffer size on the performance of CroMo-Mixup, we experiment with four buffer size options, 25, 50, 75, and 100 samples saved per task for CIFAR100-Split10. As shown in Figure 6, we observe that with smaller buffer sizes, model performance drops within the 1-2 percentage. However, even in the smaller budget of 25 samples/task, CroMo-Mixup outperforms the SOTA baseline CaSSLe+, that saves 100 samples per task, for each respective SSL baseline.

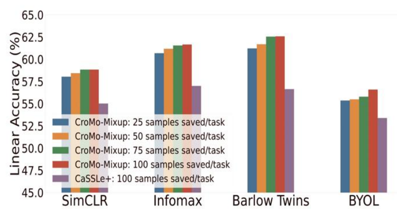


Fig. 6: Average linear accuracy performance of CIFAR100-Split10 CroMo-Mixup with varying buffer size options: 25, 50, 75, 100 samples/task. As buffer size reduces, the model performance decreases from 1-2 % at most; however, it still outperforms the SOTA baseline CaSSLe+ for each respective SSL baseline.

C.3 Out-of-Distribution Performance

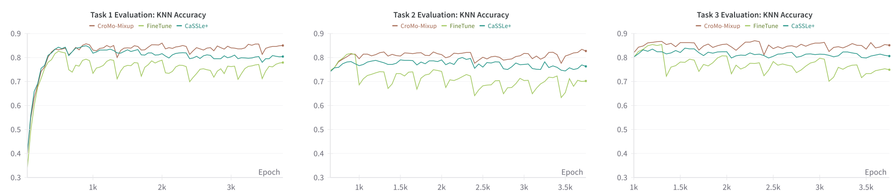
To evaluate the model's generalization performance, we test the ResNet-18 model trained with the CIFAR100-Split5 setting on two other datasets, CIFAR10 and Oxford Flower102. We compare the model trained on five different settings; offline, FineTune, Ering, CaSSLe+, and CroMo-Mixup. For all these baselines, at the end of the training, we freeze the encoder and train a linear classifier on the training dataset of CIFAR10 and Flower102 for 200 epochs. We use an SGD optimizer with a learning rate of 0.2 for the downstream task. We evaluate its performance on the test set of each respective dataset. We present the results in Table 4. The key observation from these experiments is that CroMo-Mixup outperforms CaSSL+ on both datasets. This highlights the significance of CroMo-Mixup to generalize better to the unseen distributions.

Table 4: Average Linear Accuracy Performance evaluation of CIFAR100-Split5 trained model on the test data of CIFAR10 and Flower 102 datasets

Dataset	Offine	FineTune	Ering	CaSSLe+	CroMo-Mixup
CIFAR10	82.27	71.71	75.08	77.16	80.36
Flower102	51.94	34.05	38.00	44.17	49.03

C.4 Catastrophic Forgetting Mitigation Performance

We analyze CIFAR100-Split10 as an example to compare the performance of CaSSLe+ and CroMo-Mixup to address catastrophic forgetting. Figure 7, shows the K-nearest neighbors (KNN) accuracy of the model with the progression of tasks. We use (K=200) to report the KNN accuracy.



(a) Task 1 Evaluation **(b)** Task 2 Evaluation **(c)** Task 3 Evaluation

Fig. 7: KNN Accuracy evaluation of Task 1, 2, and 3 on their respective test sets for CIFAR100-Split10 with the Barlow Twins SSL baseline.

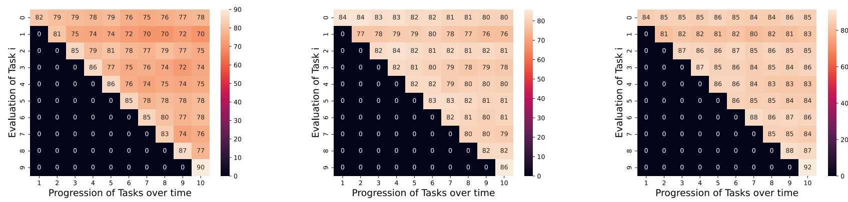
From Figure 7, we can see that finetune baseline model struggles with catastrophic forgetting. We observe a significant accuracy drop as the model learns new tasks. Though CaSSLe+ and CroMo-Mixup help to alleviate catastrophic forgetting, CroMo-Mixup outperforms CaSSLe+ by achieving 4.5%, 6%, and 4.5% higher KNN test accuracy at the end of CL training as shown in Figures 7a, 7b and 7c. For the remaining tasks, we present the confusion matrix for FineTune, CaSSLe+, and CroMo-Mixup in Figures 8a, 8b, and 8c, respectively. First we note that all these matrices are diagonal dominant. Further, for the last column of each task, CroMo-Mixup either achieves equal or higher performance than CaSSLe+, which highlights the model's ability to alleviate forgetting.

D Hyper-Parameter Configurations

In this section, we provide the hyper-parameter configurations and the SSL loss function descriptions.

D.1 Hyper-Parameter Configurations in CSSL Experiments

Table 5 presents the important hyperparameters for all SSL methods. Most hyperparameters are selected from the original code bases of these works. For CSSL



(a) FineTune **(b)** CaSSLe+ **(c)** CroMo-Mixup

Fig. 8: KNN Accuracy Performance of different tasks for CIFAR100-Split10 with the Barlow-Twins baseline. 0 here indicates the time instances when that task was not available and, therefore, is not evaluated. Comparing the last column of each task, CroMo-Mixup either achieves equal or higher accuracy than the SOTA baseline, CaSSLe+.

hyper-parameters such as epochs/task, learning rate scaling, etc, optimal hyper-parameters are found by doing a hyperparameter grid search on basic fine-tuning (FT). Subsequently, the identified optimal set of hyperparameters is uniformly applied across all CSSL methods for consistency in evaluation. Furthermore, the buffer size details are provided in Table 6. For sampling data from the memory buffer, we use a batch size of 64, which we selected by hyperparameter tuning on ER.

Table 5: Hyperparameter Settings for the CSSL experiments

CIFAR-10 / CIFAR-100 / Tiny-Imagenet	Cor-Infomax	SimCLR	BYOL	Barlow-Twins
Batch Size	512/512/256	512/512/256	256	256
Learning rate	0.1/0.1/0.5	0.6/0.6/0.3	1.0 / 1.0 / 0.3	0.3
Optimizer	SGD	SGD	LARS	LARS
Weight decay	1e-4	5e-4	1e-5	1e-4
Projection layer (dim)	64/128/64	128/128/2048	4096	2048
Prediction layer (dim, for BYOL)	-	-	4096	-
Temperature (τ)	-	0.5	-	-

D.2 Hyper-Parameter Configurations in Task Confusion Experiments

For the task confusion experiments, we use ResNet-18 encoder to train on the CIFAR100 dataset. We select the hyper-parameters following the original papers of the respective works for 100x1 and use the same settings for 10x10 experiments. Details of some of the hyper-parameters are provided in Table 7.

Table 6: CSSL Experiment Setup Details

Experiment Name	Total # of Classes	Classes per Task	Total # of Tasks	Samples saved/Task	Total # of Samples per Task
cifar10-Split2	10	5	2	500	25,000
cifar100-Split5	100	20	5	500	10,000
cifar100-Split10	100	10	10	100	5,000
tinyImageNet-Split10	200	20	10	100	50,000

Table 7: Hyper-Parameter Settings for the Task Confusion Experiments

Name	**CorInfomax**	**Barlow-Twins**	**SimCLR**	**BYOL**	**Supervised**
Optimizer	SGD	LARS	LARS	LARS	SGD
lr	0.5	0.3	0.6	1.0	0.075
epochs	1000	1000	1000	1000	200
batch size	512	256	512	256	128

D.3 SSL Loss functions

Here, we provide the details of the loss functions of the four SSL baselines on which we deployed CroMo-Mixup.

BYOL Loss Function BYOL employs a momentum encoder, where gradients are backpropagated only through the first augmentation of the data, and the second augmentation encoder network is updated by an exponential moving average (EMA). It employs an MSE-based loss function, which essentially enforces consistency between the l_2 normalized embedding vectors of z^1 and z^2 as $||q^1 - z^2||_2^2$, where $q^1 = h(z^1)$ and $h(.)$ is the predictor head.

SimCLR Loss Function SimCLR uses InfoNCE loss function which treats the second of augmentation of an image as its positive, and every other image in the mini-batch as negative. The InfoNCE loss function is calculated on the feature embeddings as follows,

$$\mathcal{L}_{\text{InfoNCE}} = -\log \frac{\exp(z_i^1, z_i^2)}{\sum_{z_j \in \eta(i)} \exp(z_i^1, z_j)} \tag{6}$$

where $\eta(i)$ contains all the negative samples of image indexed at i and all the embedding vectors are l_2 normalized.

Barlow-Twins Loss Function Barlow-twins employs a cross-correlation based loss function, as shown below,

$$\mathcal{L}_{\text{BT}} = \sum_i (1 - \mathcal{C}_{ii})^2 + \lambda \sum_i \sum_{i \neq j} \mathcal{C}_{ij}^2 \tag{7}$$

where λ is a positive hyper-parameter. The first term is invariance term that makes the feature embeddings invariant to the data augmentation, whereas second term is redundancy reduction term which reduces the redundancy in the output units. The cross-correlation $\mathcal{C}$ is computed between the feature embeddings of first and second augmentation along the batch dimension as given below,

$$\mathcal{C}_{ij} = \frac{\sum_b z^1_{b,i} z^2_{b,j}}{\sqrt{\sum_b (z^1_{b,i})^2}\sqrt{\sum_b (z^2_{b,j})^2}} \tag{8}$$

where b signifies the index for the batch samples and i, j identify the index of the vector dimension of the output embeddings.

CorInfomax Loss Function CorInfomax uses log determinant mutual information (LDMI) criterion for self-supervised learning. Its objective is essentially an estimate of LDMI between the two augmented views of model output embeddings vectors as given below,

$$\mathcal{L}_{\text{CorInfomax}} = -\log\,\det(\hat{R}^{(1)}_z[l]+\epsilon I)-\log\,\det(\hat{R}^{(2)}_z[l]+\epsilon I)+2\frac{2}{\epsilon N}||Z^{(1)}[l]-Z^{(2)}[l]||^2_F \tag{9}$$

where l identifies the batch number, $||.||_F$ is Frobenius norm. Further, $R^{(2)}_z[l]$ and $R^{(2)}_z[l]$ are auto-covariance estimates calculated as follows,

$$\hat{R}^{(1)}_z[l] = \lambda \hat{R}^{(1)}_z[l-1](1-\lambda)\frac{1}{N}\bar{Z}^{(1)}[l]\bar{Z}^{(1)}[l]^T \tag{10}$$

Likewise, $\hat{R}^{(2)}_z[l]$ is estimated based on the current and last batch statistics. Note that $\bar{Z}^{(2)}[l]$ are the mean-centralized feature embeddings, $\bar{Z}^{(2)}[l] = Z^{(2)}[l] - \mu^{(2)}[l]I^T_N$ where $\mu^{(2)}$ is the mean estimate of the current and old batches.