

Joint Diffusion Models in Continual Learning

Supplementary Material

A. Details on initial experiment - impact of the generative replay

In this section, we describe the experimental details of the initial experiment presented in Fig 1. For our classifier, we use the ResNet152 architecture with 58.3M parameters, which is comparable to 56.3M parameters of the joint diffusion model. During the experiment, we first train both the classifier and joint diffusion model on the CIFAR10 dataset and save both models. We then use the trained joint diffusion to sample a synthetic dataset - notably, both images and labels are generated by the joint diffusion model. Then, in the first variant of the experiment, we continue training both saved models on this synthetic dataset. In a second variant, we additionally incorporate a knowledge distillation loss during the continued training. For the joint diffusion model, this involves applying the distillation approach detailed in our method, leveraging the saved model trained on the original CIFAR10 data. As for the ResNet152, we utilize the saved classifier trained on the original data and apply only the classification component of the distillation loss of our method.

B. Main experiments - implementation details

We train diffusion using linear noise schedulers over 1000 steps and use DDIM with 250 steps to generate replay samples. In particular, we generate 1000, 400, and 1040 samples per class in CIFAR10, CIFAR100, and ImageNet100 respectively. For the initial local model, we train for 50,000 steps on CIFAR10, 100,000 steps on CIFAR100, and 120,000 steps on ImageNet100. Subsequent tasks are trained for 30,000 steps on CIFAR10, 70,000 steps on CIFAR100, and 90,000 steps on ImageNet100. Additionally, following [12], for the first 10000 steps of every *global* model training, we do not calculate the classification loss and optimize only the generative objective. We use a single AdamW optimizer with a learning rate set to 0.0002 and a batch size of 256. For all experiments, we set the classification loss scales α and α_{KD} to 0.001, and the β hyperparameter that scales joint-diffusion loss to 0.01.

We define three sets of data augmentations. The first one consists of flipping, resized cropping, and normalization to range $[-1, 1]$, and is used to train the denoising model. The second, *strong* augmentation set includes flipping, cropping, the RandAugment [8] augmentation policy, and normalization to range $[-1, 1]$. The third and final, *weak* augmentation set consists of flipping, cropping, and normalization to range $[-1, 1]$. We use the *strong* augmentation set to perturb the images for the classifier in the supervised setups.

Additionally, following [63], we use our *strong* and *weak* augmentation sets to compute the semi-supervised loss. We also set the τ threshold to 0.95 and we set the supervised-to-unsupervised batch size ratio to 7, which corresponds to a supervised batch size of 32 and an unsupervised batch size of 224.

C. Stability of the JDCL’s representation space in an incremental setup

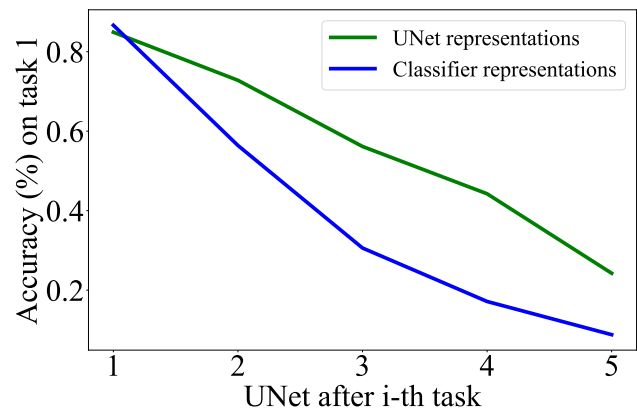


Figure 7. Accuracy of the kNN-classifier trained on representations provided by the model after the first task of class-incremental CIFAR100 setup with 5 tasks

One of the most important features of our methods is its ability to learn a meaningful representation space shared by the generative and discriminative parts of the model. To better understand how this representation space changes during the class-incremental training, we propose to measure the drift of representations, with the following experiment on a 5-task CIFAR100 setup. We first extract data representations from two parts of our UNet model, the H-space inside of the generative part of our model, and the penultimate layer of the classifier head. We train a kNN-classifier on top of those representations provided by the model after the first task. Then, we evaluate how the accuracy of this classifier changes when the representations for data from the first task are extracted with models trained incrementally on later tasks. We present the results in Fig. 7. We observe that the representations extracted from the UNet change significantly slower than those from the classifier. This means that representations that are not used for classification drift significantly slower than those that are classifier-specific, which highlights the benefits of joint modeling in a continual setup.

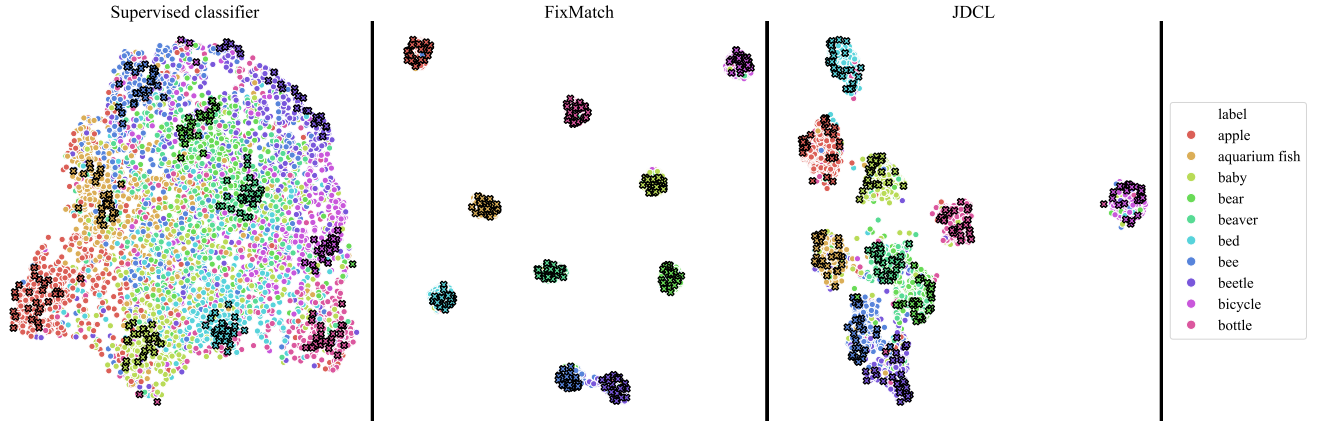


Figure 8. Visualization of the representation spaces (umap) of different methods trained on a partially labeled dataset. Labeled data samples are marked with an "X", while unlabeled data points are marked with circles. All the models were trained on the first 10 classes of CIFAR100 with 5% of the data being labeled.

Table 6. Comparison of our JDCL with generative rehearsal methods including feature replay techniques that we mark in gray color. Results of competing methods from [9].

METHOD	AVERAGE ACCURACY \bar{A}_T (\uparrow)				AVERAGE FORGETTING \bar{F}_T (\downarrow)			
	CIFAR-10	CIFAR-100	IMAGENET100		CIFAR-10	CIFAR-100	IMAGENET100	
	$T = 5$	$T = 5$	$T = 10$	$T = 5$	$T = 5$	$T = 5$	$T = 10$	$T = 5$
JOINT	93.14 \pm 0.16	72.32 \pm 0.24	66.85 \pm 2.25	-	-	-	-	-
CONTINUAL JOINT	86.41 \pm 0.32	73.07 \pm 0.01	64.15 \pm 0.98	50.59 \pm 0.35	2.90 \pm 0.08	7.80 \pm 0.55	6.67 \pm 0.36	12.28 \pm 0.07
FINE-TUNING	18.95 \pm 0.20	16.92 \pm 0.03	9.12 \pm 0.04	13.49 \pm 0.18	94.65 \pm 0.17	80.75 \pm 0.22	87.67 \pm 0.07	64.93 \pm 0.00
DGR VAE	28.23 \pm 3.84	19.66 \pm 0.27	10.04 \pm 0.17	9.54 \pm 0.26	57.21 \pm 9.82	42.10 \pm 1.40	60.31 \pm 4.80	40.46 \pm 0.91
DGR+DISTILL	27.83 \pm 1.20	21.38 \pm 0.61	13.94 \pm 0.13	11.77 \pm 0.47	43.43 \pm 2.60	29.30 \pm 0.40	21.15 \pm 1.30	41.17 \pm 0.43
RTF	30.36 \pm 1.40	17.45 \pm 0.28	12.80 \pm 0.78	8.03 \pm 0.05	51.77 \pm 1.00	47.68 \pm 0.80	45.21 \pm 5.80	41.2 \pm 0.20
MERGAN	51.65 \pm 0.40	9.65 \pm 0.14	12.34 \pm 0.15	-	-	-	-	-
BIR	36.41 \pm 0.82	21.75 \pm 0.08	15.26 \pm 0.49	8.63 \pm 0.19	65.28 \pm 1.27	48.38 \pm 0.44	53.08 \pm 0.75	40.99 \pm 0.36
GFR	26.70 \pm 1.90	34.80 \pm 0.26	21.90 \pm 0.14	32.95 \pm 0.35	49.29 \pm 6.03	19.16 \pm 0.55	17.44 \pm 2.20	20.37 \pm 1.47
DDGR	43.69 \pm 2.60	28.11 \pm 2.58	15.99 \pm 1.08	25.59 \pm 2.29	62.51 \pm 3.84	60.62 \pm 2.13	74.70 \pm 1.79	49.52 \pm 2.52
DGR DIFFUSION	59.00 \pm 0.57	28.25 \pm 0.22	15.90 \pm 1.01	23.92 \pm 0.92	40.38 \pm 0.32	68.70 \pm 0.65	80.38 \pm 1.34	54.44 \pm 0.14
GUIDE	64.47 \pm 0.45	41.66 \pm 0.40	26.13 \pm 0.29	39.07 \pm 1.37	24.84 \pm 0.05	44.30 \pm 1.10	60.54 \pm 0.82	27.60 \pm 3.28
JDCL	83.69 \pm 1.44	47.95 \pm 0.61	29.04 \pm 0.41	54.53 \pm 2.15	11.13 \pm 3.27	37.20 \pm 0.75	59.55 \pm 0.66	19.51 \pm 6.27

D. Representation space in a semi-supervised setup

In this work, we introduce a new method of training joint diffusion model in a semi-supervised setup. In this section, we show that this approach allows our model to learn shared representations that are meaningful for both generative and discriminative objectives. To that end, we perform a visual comparison of the representation spaces in Fig. 8. We compare our method with a state-of-the-art semi-supervised method FixMatch [63], and a baseline classifier model that we train only on the labeled data. To extract the representations, we average pool the UNet representations into a single vector in our approach, while for the other two models, we take the output of the second to last layer. We observe that our approach is able to group the labeled and unlabeled data from the same class into similar regions of the embedding space shared between the classifier and diffusion model, similar to the state-of-the-art FixMatch approach.

E. Additional results - average forgetting

To further evaluate our approach, we use another metric commonly used in continual learning - average forgetting, defined as in [6]. In Tab. 6 we present the results of JDCL alongside competing generative replay techniques. Our method outperforms all other approaches on both CIFAR10 and ImageNet100. For CIFAR100 benchmarks, although JDCL outperforms other methods on the average accuracy metric, we observe higher forgetting when compared to the GFR. However, as discussed in Sec. 5.2 and shown in Fig. 3, this can be attributed to the limited plasticity of the GFR that significantly restricts its model’s update. This approach, while admittedly leading to lower average forgetting, results in worse plasticity, and consequently significantly worse final performance.

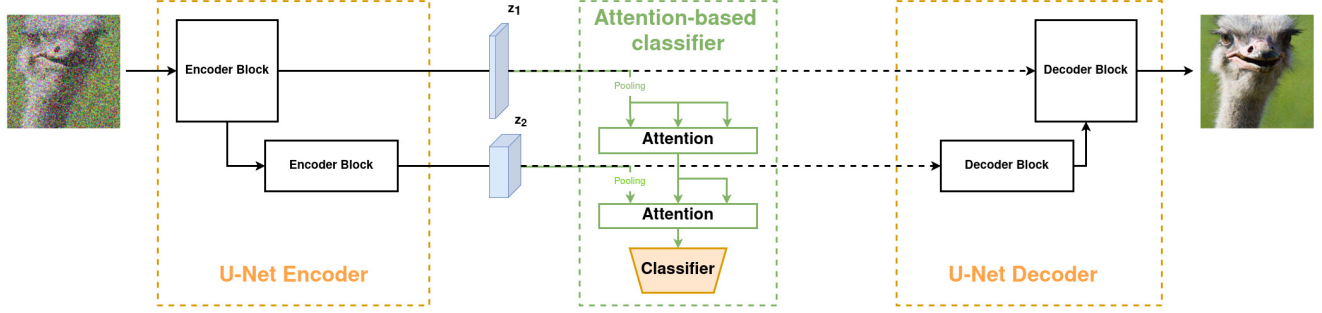


Figure 9. Visualization of the Joint Diffusion with attention-based classifier. The classification method processes the representations from the shallowest to the deepest levels of UNet architecture using the attention mechanism to create a feature vector.

Table 7. Comparison of the classification accuracy and FID scores of the Joint Diffusion model trained with the attention-based and with the pool-based classifier.

METHOD	CIFAR-10			CIFAR-100		
	ALL LABELS	1000 LABELS	250 LABELS	ALL LABELS	10000 LABELS	2500 LABELS
JD: ATTENTION - ACC	95.7%	93.9%	90.9%	78.3%	72.9%	63.4%
JD: POOLING - ACC	96.4%	92.1%	86.6%	77.6%	69.9%	60.8%
JD: ATTENTION - FID	15.9	28.2	28.5	23.1	-	-
JD: POOLING - FID	7.9	27.0	27.4	17.4	-	-

Table 8. Comparison of the FID scores of JDCL and Deep Generative Replay with standard DDPM after each task of the CIFAR100/5 setup.

METHOD	TASK				
	1	2	3	4	5
DGR/GUIDE	19.08	29.52	39.11	40.75	50.36
JDCL	18.20	28.06	29.73	31.39	39.93

F. Evaluation of generative qualities

To further evaluate the generative part of our method, we perform additional experiments. In Tab. 8, we compare the FID scores after every task of the CIFAR100/5 setup. Our method outperforms DGR (and GUIDE) after each task. To better understand the advantage of our approach, in Tab. 9 we calculate the equivalents of Forward and Backward Transfer (as in DER [79] for the FID metric). We observe that our method is significantly more stable and slightly less plastic, which is consistent with the results on the discriminative task. Nevertheless, JDCL’s overall FID performance is much higher, approaching the IID upper bound equal to 29.7.

G. Other ways to perform classification in the Joint Diffusion model

We extensively explored the design of the classifier during our preliminary studies. Here, we include a comparison of

Table 9. Forward and backward FID transfer of JDCL and Deep Generative Replay on CIFAR100/5.

METHOD	FORWARD TRANSFER (\downarrow)	BACKWARD TRANSFER (\downarrow)
DGR	19.80	10.22
JDCL	22.80	7.80

the selected pooling method with one of those alternative approaches - an attention-based classifier. Instead of pooling features into a single vector, this method applies a chain of attention blocks over the UNet’s h-representations. Each block takes the corresponding layer’s tensor and the previous block’s output to compute inter-level attention, merging them into a single vector. Consequently, the classifier better exploits the representations and achieves higher accuracy on most tested setups, as shown in Tab. 7. However, training with the attention-based approach degrades the quality of the generations, making it less suitable for CL.

H. Additional Backward and Forward Transfer metrics

In Tab. 10, we present the additional Forward and Backward Transfer metrics for JDCL, GUIDE, and GRF on CIFAR100/5 setup. The calculated metrics support our claims from Fig. 3. JDCL significantly outperforms GUIDE on Backward and GRF on Forward Transfer.

Table 10. Comparison of Forward and Backward Transfer of JDCL, GRF and GUIDE on CIFAR100/5 setup.

METHOD	FORWARD TRANSFER (\uparrow)	BACKWARD TRANSFER (\uparrow)
GUIDE	-15.72	-26.86
GRF	-49.99	-10.00
JDCL	-16.61	-14.83

Table 11. Comparison of the training diffusion loss of the jointly and separately trained diffusion model on the first task of CIFAR10/5.

Runtime	Joint diffusion	Separate Diffusion
1h	0.01785	0.01982
2h	0.01741	0.01844
3h	0.01691	0.01723

I. Details on semi-supervised scenario

All experiments follow offline CL protocols, while our semi-supervised scenario builds on the NNCSL [26], designed specifically for SSL in CL. In the paper authors adapt various offline approaches to the SSL framework, and we adhere to their evaluation protocol. Due to an additional diffusion model, we extend JDCL training to 60k batch updates for CIFAR10.

J. Training complexity - discussion

While joint training increases the complexity of a single DM training step, it eliminates the need to train a separate classifier. Consequently, our joint model has fewer parameters (56.3 M) than the separate classifier and DM combined (64.7 M). Additionally, because of the synergy of the generative and discriminative parts, our method converges faster than the standard DM. We show that in Tab. 11, where we compare training loss on the first task of CIFAR10/5 of the 2 methods after the same runtime.

K. Sensitivity of performance with respect to loss terms scales

For all experiments, we use exactly the same classification loss scales equal to 0.001 and non-knowledge-distillation loss scale β equal to 0.01. However, to present a full picture, we present the influence of different β values on the final score on CIFAR10/5. Notably, the majority of results are within statistical errors:

β	0	0.001	0.005	0.01	0.05	0.1	0.5	1
ACCURACY/GAIN	-0.59	-1.37	-0.37	83.69\pm1.44	+0.32	-1.16	-6.79	-10.41