Introspective GAN: Learning to Grow a GAN for Incremental Generation and Classification Supplementary Material

Chen He[a,b], Ruiping Wang[a,b,*], Shiguang Shan[a,b], Xilin Chen[a,b]

[a] *Key Laboratory of Intelligent Information Processing of Chinese Academy of Sciences (CAS), Institute of Computing Technology, CAS, Beijing, 100190, China*
[b] *University of Chinese Academy of Sciences, Beijing, 100049, China*

1. Introduction

In this supplementary material, we provide more discussions with some related works (Sec. 2 in the main paper), the analysis of the prototype-based classifier via a Bayesian approach (Sec. 3.3), the comparison of the adopted benchmark with existing ones in incremental generation or classification (Sec. 4.1 in the main paper), implementation details (Sec. 4.1 in the main paper), the additional result curves on MNIST and ImageNet-Dogs (Sec. 4.2 in the main paper), the generated images over time (caption in Fig. 7), the numbers of generated samples of each class of DGR (Sec. 4.5 in the main paper), hyper-parameter analysis (multiple sections in the main paper), memory requirements and running time comparisons (Sec. 4.1), memory comparison for GANs and real samples (Sec. 4.6), and the class orders (Table 2 in the main paper) respectively.

2. More Discussions with Some Related Works

Their similarities and differences with several works [1, 2, 3] are as follows:

(1) This work [1] thoroughly compares different incremental generation methods and different choices of the generators (e.g. GAN and VAE) on MNIST,

*Corresponding author
Email address: wangruiping@ict.ac.cn (Ruiping Wang)

F-MNIST, and CIFAR-10 (only with best performing CL strategy), whereas we are more interested in a joint incremental generation and classification task and compare the methods for this task. Moreover, the latest evaluated method is DGR which was proposed in 2017, where the latest evaluated method is DGM which was proposed in 2019.

(2) This work [2] mainly focuses on Class Incremental Learning (classification) and the generators (i.e. VAEs) used in this work are for generative replay to boost the classification performance, where our work focuses on the joint incremental generation and classification task and validate the mutual benefits of generation and classification in incremental scenarios.

(3) This work [3] provides an interesting idea by pre-allocating binary codes (i.e. hash codes) for image samples and leveraging autoencoders to reconstruct these samples via the binary codes above. The work mainly revolves around incremental classification and focuses on minimizing the memory usage in memory replay by using binary codes (since they require less memory compared with real vectors), while our work mainly focuses on the mutual benefits of incremental generation and classification.

Diffusion models are effective and popular in recent years. Readers may be curious about why we still use GANs instead of diffusion models. Here we summarize the limitations and strengths of GANs over diffusion models, and share our thoughouts over these two models.

Limitations: (1) The generated images of diffusion models are of higher quality compared with those of GANs judging from the fewer artifacts in the generated images or FIDs in recent papers [4, 5]. (2) The GAN models are less stable for training due to the adversarial objective, and often suffer from the mode collapse problem [6]. In contrast, diffusion models have a stable training process and provide more diversity because they are likelihood-based [5].

Strengths: (1) The forward pass in GAN is generally quicker, whereas the diffusion models are slower at sampling time due to the use of multiple denoising steps [7]. There are diffusion models that use a single-step sampling, but the samples are not yet competitive with GANs [8]. (2) The latent space of GANs contains subspaces associated with visual attributes, thus by changing the value of certain feature we can edit the image (image editing applications) [9]. As for the diffusion models, the latents are usually modeled to have the same dimensionality as the image, resulting in less semantic information in the latent space. In short, the latent space of diffusion models has been explored much less than in the case of GANs [5].

The aforementioned strengths and weaknesses are also confirmed by our experimental results with DDGR [10]: DDGR generates images of better quality, which leads to much higher classification accuracy. However, the sampling speed of DDGR is 1.48s per image, while the sampling speed of IntroGAN is only about 0.002s (over 600 times faster). The training time of DDGR is 3 hours in the first class increment of Fashion-MNIST, while that of IntroGAN is 9 minutes (20 times faster). The slowness of training and sampling will restrict the actual use in incremental scenarios (e.g. mobile devices).

Judging from the limitations and strengths above, we can find that these two generative models are trade-offs between performance and efficiency, which are suitable for different scenarios currently. Going back to our work, we select GANs simply due to its dominance in 2020s, but the generator can also be replaced with other generative models including the diffusion model. Note that the DDGR is more like a diffusion-model based version of DGR. Our main focus is the mutual benefit of generation and classification in incremental scenarios. Choosing which type of generative models is orthogonal to our motivation. Another thing to note is that in the "Few-Shot Learning Methods" part of Sec. 3.5, we write that we can either learn class distributions (ours) or perturbation. We think that with diffusion models, we are able to learn perturbation by gradually adding/removing noises to/from the image, and we can finally get the class prototype in the image space. This interesting idea is left for future works.

3. Prototype-based Classifier from the Bayesian Perspective

In Sec. 3.3 of the main paper, we mentioned that the prototype-based classifier using *relative prototype* has a Bayesian background and here we will explain why. Assuming that the features of each class samples obey a multivariate Gaussian distribution (dimension d), the probability density function is:

$$p(x|c) = \frac{1}{(2\pi)^{\frac{1}{2}}|\Sigma_c|^{\frac{d}{2}}} \exp\{-\frac{1}{2}(F(x)-\mu_c)^T\Sigma_c^{-1}(F(x)-\mu_c)\} \tag{1}$$

According to the Bayes' theorem, we have:

$$p(c|x) = \frac{p(x|c)p(c)}{p(x)} = \frac{p(x|c)p(c)}{\Sigma_{i=1}^{K} p(x|i)p(i)} \tag{2}$$

Assuming that each class has equal prior probability $p(i)$ and the covariance matrix is $\sigma^2 I$ where I is an identity matrix, the above equation becomes:

$$p(c|x) = \frac{\exp(-\frac{1}{2\sigma^2}\|F(x)-\mu_c\|_2^2)}{\Sigma_{i=1}^{K}\exp(-\frac{1}{2\sigma^2}\|F(x)-\mu_i\|_2^2)} \tag{3}$$

In the main text, we use $p^{(c)}$ to estimate μ_c and γ to substitute $\frac{1}{2\sigma^2}$. The above equation becomes:

$$p(c|x) = \frac{\exp(-\gamma\|F(x)-p^{(c)}\|_2^2)}{\Sigma_{i=1}^{K}\exp(-\gamma\|F(x)-p^{(i)}\|_2^2)} \tag{4}$$

It is exactly Eq. 7 when L2 distance is used in the main text, which demonstrates the relationship between Bayesian probability and prototype-based classifier.

4. Comparisons of Incremental Learning Benchmarks

We refer to the existing benchmarks in incremental generation or classification to design the joint task. Table 1 lists the datasets and settings adopted in previous studies of incremental generation, incremental classification and our joint task. The numbers of classes used by incremental generation methods are generally no more than 10. It is because incremental generation is harder and less explored compared to incremental classification. Therefore we make a compromise and mainly refer to settings for incremental generation when designing the new benchmarks. From Table 1 it can be seen that the numbers of classes and training samples for IntroGAN are at the same level with previous incremental generation approaches, indicating that the new benchmarks are reasonable.

Another compromise is on how many classes to be added in each training session. Since adding one class at a time is not a natural setting for incremental classification (when there is only one class, training a classifier is meaningless). Besides, almost all incremental classification methods start with adding two classes at a time. Therefore, we also start from adding two classes at a time as shown in Table 2 in the main paper. Adding two classes at a time is a difficult setting because it leads to more training sessions which incurs much severe catastrophic forgetting (this phenomenon can be seen in other papers like [11, 12]). Thus, we perform the challenging two-class adding experiments on relatively easy datasets MNIST, Fashion-MNIST and SVHN. For ImageNet-Dogs, we add ten classes at a time because most methods perform less satisfactory even under this setting and it is thus not necessary to further increase the difficulty by using the two-class adding scenario. Also, adding two classes at a time on ImageNet-Dogs needs much more time for training (15 training sessions).

5. Implementation Details

All methods adopt an Adam optimizer [13] with a base learnings rate 2×10^{-4} and train for 10,000 iterations with a batch size of 100. For MNIST [14], Fashion-MNIST [15] and SVHN [16], two parameters (β_1, β_2) of the Adam optimizer are set to (0.5, 0.999). The discriminator/classifier network is LeNet-like with three convolutional layers and one fully connected layer; the generator is roughly the reversed version of the discriminator/classifier inspired by the implementation of [17]. For ImageNet-Dogs [18] (ImageNet-Dogs is a special down-sampled Stanford Dogs [19]), (β_1, β_2) are set to (0, 0.9) and the batch size is 64. The

Dataset	Generation			Classification		Joint
	DGM	**MeRGAN**	**DGR**	**iCaRL**	**ESGR**	**IntroGAN**
MNIST	10,60K	10,60K	10,60K	-	-	10,60K
F-MNIST	-	-	-	-	-	10,60K
SVHN	10,73K	10,73K	10,73K	-	-	10,73K
C-10	10,50K	-	-	-	-	-
C-100	-	-	-	100,50K	100,50K	-
ImageNet	30,39K/ 50,65K	-	-	100,130K/ 1000,1300K	120,156K	30,39K
LSUN	-	4,400K	-	-	-	-

Table 1: The number of classes and training samples of the datasets chosen by different methods (separated by the comma). F-MNIST is short for Fashion-MNIST. C-10 and C-100 are short for CIFAR-10 and CIFAR-100. Note that ESGR is classified into classification method because it is oriented for classification only.

network is ResNet-like with four ResBlocks for the generator and five for the discriminator/classifier with spectral normalization [20].

Joint Training. The model is just a conventional inner-product based softmax classifier. For each class increment, all training samples from the seen classes are added for training and the fully connected layer is initialized randomly. The other hyper-parameters are almost the same with the implementations of other methods.

Fine-tuning. It is almost the same with the settings of Joint Training above except: (1) for each class increment only new class samples are used for training; (2) the improved initialization technique is used.

IntroGAN (Introspective GAN). We assign 20 prototypes for each class ($M = 20$). The temperature γ controls the smoothness of output probabilities to make them neither too close nor distant. Since the squared L2 distance in Eq. 7 is usually big ($\approx 10^2$ in practice), a smaller γ should be better for optimization and it is set to 0.01 by default (for the experiment in Sec. 4.5 it is set to 0.1).

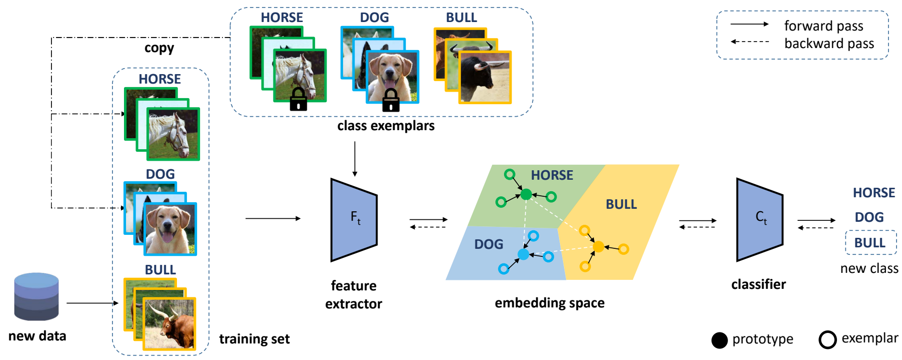

Figure 1: Schematic illustration of the variant of IntroGAN named IntroNet similar to Fig. 4 in the main paper. Compared to IntroGAN, the discriminator and the generator are removed, and only a prototype-based classifier is left.

Our experience is to let the input of the $exp(\cdot)$ in Eq. 7 have a magnitude of around 1. The weighting factors α and β in the classification loss are set to 0.1 and 1 respectively inspired by the original implementation of AC-GAN [21]. The class prototypes are updated at a certain interval (e.g. 2,000 iterations) for *feature k-means* and stay fixed for feature invariant settings like *random* and *image k-means*.

IntroNet. For better understanding, we draw an illustration diagram similar to Fig. 4 in the main paper to show the framework of IntroNet (Fig. 1 here). As illustrated in Sec. 4.4 in the main paper, we remove the GAN out of the original IntroGAN framework. Specifically, we remove the discriminator and the generator, thus there is only one prototype-based classifier left. Since there are no generated samples of old classes to be replayed anymore, the training set at time t becomes $\{X_{exem}^{(1)}, \ldots, X_{exem}^{(t-1)}, X_{train}^{(t)}\}$, which is usually imbalanced and we duplicate the exemplars of old classes for multiple times to make each class equally sampled. To make fair comparison, we use exactly the same exemplars selected in IntroGAN.

MeRGAN-JTR (Memory Replay GAN-Joint Training [22]). As illustrated in the main paper, the official implementation by the original author of MeRGAN-JTR [22] performs badly in classification (accuracy for Task 1-5 on

MNIST: 99%, 49%, 33%, 25%, 21%). The less satisfactory result is not because the code is run wrongly, but due to the fact that the original method does not consider using the classifier of AC-GAN for classification nor try to optimize this performance. However, one tiny modification can bring up the performance, which is to create a balanced training set combining old replayed samples and new real samples as in the training process of IntroGAN (mentioned in Sec 3.4 in the main paper). Thus, all results of MeRGAN-JTR reported in our paper are based on this modified version with improved learning techniques. Similar to IntroGAN, the weighting factors in the classification loss are set to 0.1 and 1 for generated data and real data respectively inspired by the implementation of AC-GAN in [17].

DGR (Deep Generative Replay [23]). The official code of DGR is not released and we implement it ourselves. The ratio r represents the desired importance of a new task compared to the older tasks. Since the authors didn't offer experience on how to choose the optimal r, we empirically set r to 0.5 which is also used in the most popular Github non-official implementation of DGR[1]. The generator and the solver both use an Adam optimizer with a base learning rate of 2×10^{-4} and are trained for 10,000 iterations.

iCaRL [11]. We use the code released by the author. The original implementation of iCaRL uses a fixed memory bound to store exemplars of all classes (the more classes, the less exemplars for each class). To match our experimental setups, it is changed to assign M exemplars for each class (note that it is not our invention and other works also have this setting [18, 24, 12]). Also, the original codes of iCaRL trains for a fixed number of epochs instead of iterations. To make fair comparison with others, we convert iCaRL to iteration-based training and Adam optimizer (we tune the best learning rate) on MNIST, Fashion-MNIST and SVHN. For ImageNet-Dogs, we empirically use 60 epochs and stochastic gradient descent with momentum as in the original code of iCaRL (their best settings). To show the result of iCaRL in the iteration-based curve like Fig. 3,

[1] https://github.com/kuc2477/pytorch-deep-generative-replay

we only evaluate the final model of each task (in this situation, iterations on x-axis does not mean anything).

LwF (Learning without Forgetting [25]). We use the version implemented by iCaRL (the iCaRL paper names it LwF.MC), which is a little different from the original LwF in that: softmax is changed to sigmoid for classifcation; the prototypes are also added to the training set for training. On ImageNet-Dogs, we also adopt the original epoch-based training which is the same as iCaRL's mentioned above.

DGM [26]. The original codes and hyper-parameters from the authors are used. We only change the setting from adding one class to adding two classes at a time to match our experimental setups. Note that DGM also uses a fixed number of epochs instead iterations, so only the performance of the final model of each task is shown on the result curves (similar to iCaRL mentioned above). The authors offer codes on MNIST, which can be easily adapted to Fashion-MNIST. As for ImageNet-Dogs, we use the official code of DGM on ImageNet and only changed the classes to be learned (i.e. keeping their recommended hyper-parameters).

DDGR. The original codes and basically the same hyper-parameters from the authors are used. We add data loaders for MNIST, Fasion-MNIST, and SVHN. For ImageNet-Dogs, the obtained accuracy using the recommended hyperparameters by the authors is almost similar as random guess (e.g. for ten classes, the accuracy is about 10%), which we think is more likely an inappropriate choice of the hyperparameters instead of the problem of the method itself. Thus, we do not show the results on ImageNet-Dogs in Table 3 in the main paper and fill the results with blanks ("-") in the table.

6. Additional Results

6.1. Result Curves on MNIST and ImageNet-Dogs

To save space, we only show the overall performance of different methods on MNIST and ImageNet-Dogs measured by TA-ACC/FID in the main paper.

Here we add the corresponding ACC and FID curves on MNIST and ImageNet-Dogs similar to those on Fashion-MNIST and SVHN (shown here in Fig. 2 and 3). It can be observed that IntroGAN still takes the lead in ACC, indicating its strong discriminative power endowed by the prototype-based classification.

On MNIST (Fig. 2), the performances of different methods are quite similar to those on Fashion-MNIST in the main paper except that MeRGAN-JTR and IntroGAN are on a par for both classification and generation. We attribute the satisfactory performances of MeRGAN-JTR and IntroGAN to the superiority of the end-to-end training of the generator and classifier which is in essence endowed by AC-GAN. Since MNIST is an easy dataset, those two methods have nearly identical performance compared to others.

On ImageNet-Dogs (Fig. 3), IntroGAN demonstrates superior performance for incremental classification, which is even better than *Upperbound*. The reason is probably due to the robustness of the prototype-based classifier which is also analyzed in Sec. 4.2 and Sec. 4.3. iCaRL is lower than LwF and it is probably because iCaRL is sensitive to the choice of the feature extractor, which is already explained in Sec. 4.2 in the main paper. As for generation, the performance of IntroGAN is in the middle of the three. The reason might be that the dataset is hard but IntroGAN has a limited capacity. IntroGAN sacrifices some generation performance for a much better classification performance.

6.2. Generated Images Over Time

In this subsection, we show the generated images over time for IntroGAN (Fig. 4) and MeRGAN (Fig. 5) on MNIST, Fashion-MNIST, SVHN and ImageNet-Dogs. From these results, it can be observed that both methods are able to generate the old classes pretty well in incremental scenarios, which is consistent with the quantitative results via FIDs in the main paper. As for DGR, since it uses a unsupervised GAN, we cannot fix the noise and label to observe the generated images over time. Thus, we do not show the generated images of DGR.

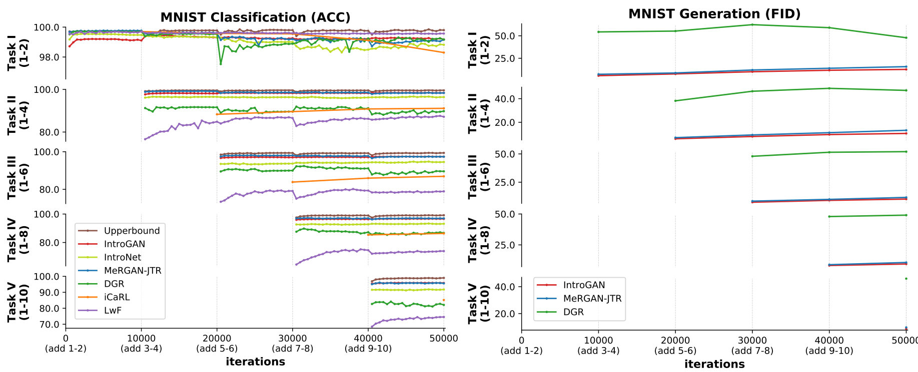

Figure 2: The ACC and FID curves of different methods on MNIST. Five sub-figures vertically indicate five different tasks. Note that for FID, the lower, the better.

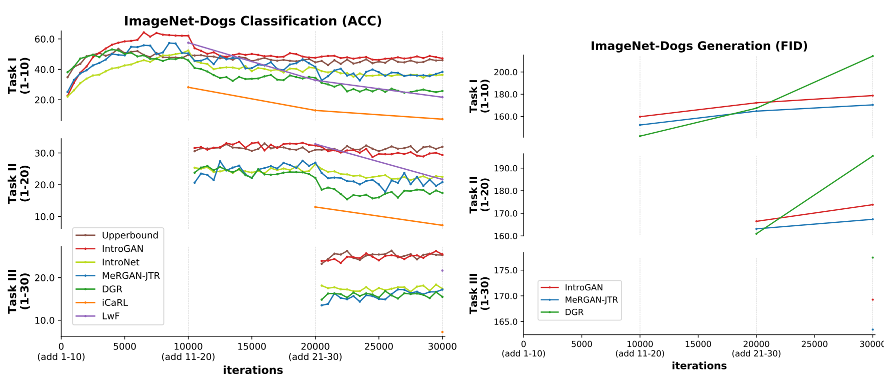

Figure 3: The ACC and FID curves of different methods on ImageNet-Dogs. Three sub-figures vertically indicate three different tasks. Note that for FID, the lower, the better.

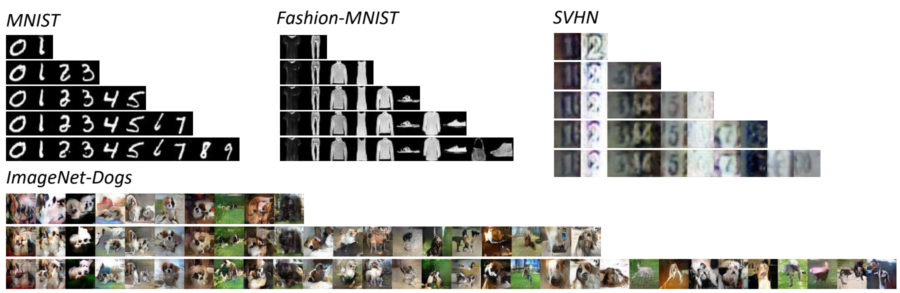

Figure 4: The generated images of IntroGAN over time on four datasets. Each row of a dataset represents a class increment. For each column of the same dataset, the used noise vector is the same.

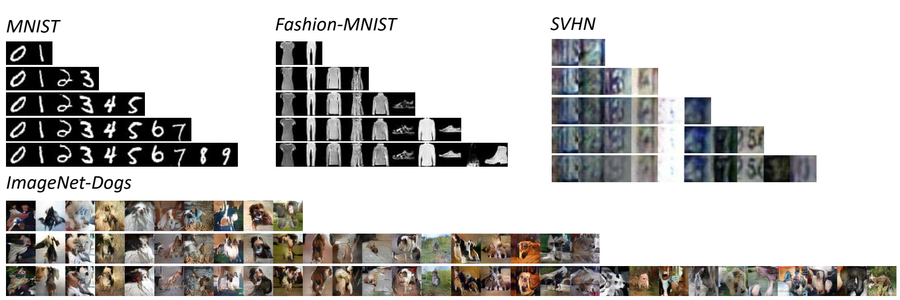

Figure 5: The generated images of MeRGAN over time on four datasets. The organization of this figure is the same as Fig. 4.

MNIST			Fashion-MNIST			SVHN		
Class	Number	Ratio	Class	Number	Ratio	Class	Number	Ratio
‘0’	0	0.00%	T-shirt	20299	20.30%	‘0’	1102	1.10%
‘1’	97128	97.13%	Trouser	50905	50.91%	‘1’	45652	45.65%
‘2’	54	0.05%	Pullover	5770	5.77%	‘2’	33928	33.93%
‘3’	605	0.61%	Dress	10024	10.02%	‘3’	6677	6.68%
‘4’	34	0.03%	Coat	5714	5.71%	‘4’	4415	4.42%
‘5’	217	0.22%	Sandal	57	0.06%	‘5’	2566	2.57%
‘6’	115	0.12%	Shirt	6932	6.93%	‘6’	1972	1.97%
‘7’	527	0.53%	Sneaker	1	0.00%	‘7’	1633	1.63%
‘8’	485	0.49%	Bag	281	0.28%	‘8’	977	0.98%
‘9’	835	0.84%	Ankle boot	17	0.02%	‘9’	1078	1.08%

Table 2: The number of the generated samples for each class of DGR on three datasets (100,000 in total for each dataset). The ratio is obtained by dividing the number of samples by the total number of generated samples 100,000.

6.3. Numbers of Generated Images of DGR

In Sec. 4.4 of the main paper, we show the pie charts which depicts the portions of generated images of all classes using DGR. The full statistics are shown in Table 6.1, which provides a more quantitative view of the pie charts in the main paper. From the table it can be seen that the generated samples of DGR are highly imbalanced, especially on MNIST DGR even fails to generate digit ‘0’.

6.4. Hyper-parameter Analysis

Exemplar Selection Strategy. We evaluate the following settings: (1) select randomly (*random*); (2) select M samples closest to M cluster centers respectively in the image space via K-means (*image k-means*); (3) same as (2) except that we cluster in the feature space (*feature k-means*). From Table 3, the conclusion is that *image k-means* is slightly better than *random*, and they both outperform *feature k-means* by a large margin. The reason for the unsatisfactory result of *feature k-means* is that it is performed in the feature space which relies on the current classification task. Incremental learning algorithms, however, need to have far-sight and store information that is important for the future tasks as well: *image k-means* and *random* are more likely to achieve this goal. Other papers also have similar conclusions that specially designed

Exem. Selection	MNIST		Fashion-MNIST		SVHN	
	ACC	**FID**	**ACC**	**FID**	**ACC**	**FID**
Image k-means	**97.57**	**8.63**	**88.22**	**25.09**	**77.55**	105.88
Feature k-means	95.07	110.23	86.93	26.17	71.53	95.07
Random	97.41	8.90	88.17	25.45	77.26	**91.64**

Table 3: The results of IntroGAN using different exemplar selection strategies on (Fashion-)MNIST and SVHN. The best score is highlighted in **bold**. Exem.=Exemplar.

exemplar selection strategy does not have substantial superiority over *random* selection [27, 24]. Due to the simpleness and effectiveness of *random* selection, we use it by default.

Number of Exemplars. we present the analysis of the number of exemplars for each class M in Table 6.4. Here we simply use the *min* selection function. However, the conclusion will not change by using different selection functions. The finding is that more exemplars yield better classification performance (in MNIST, there is not much difference in ACC because the dataset is rather easy), but the benefits to generation is less obvious. The reason might be that we adopt a prototype-based classifier, but the generator does not closely rely on prototypes. Thus, the performance gain in classification is more obvious and reasonable, since a larger M will make the estimation of the real class centroid in the feature space more accurate as explained Sec. 3.2, and the dual use of the exemplars as training samples mentioned in Sec. 3.4 also benefit the classification performance.

Effectiveness of Improved Initialization. To demonstrate its effectiveness, we compare another two trivial settings: *Random initialization (all layers)*: initialize all parameters randomly when entering a new training session; *Random initialization (exclusive layers)*: only initialize parameters of the exclusive layers randomly and keep other shared layers untouched. The results of IntroGAN, DGR, MeRGAN with different initialization strategies on (Fashion-)MNIST and

M	MNIST		Fashion-MNIST		SVHN	
	ACC	**FID**	**ACC**	**FID**	**ACC**	**FID**
1	97.21	9.29	84.06	25.82	50.73	107.18
5	96.01	8.92	86.26	25.25	66.27	99.33
20	96.37	16.00	88.25	25.59	74.64	106.36
100	96.57	8.41	89.42	25.25	79.62	102.04

Table 4: The results of IntroGAN via different number of exemplars M on three datasets.

SVHN are shown in Table 5. For most cases, improved initialization can offer better performance for all these methods and the reason is that it can encourage more knowledge transfer from old classes to new classes as elaborated in Sec. 3.4. Therefore, we use *improved initialization* by default.

Prototype-based Classification Strategy for Testing. Different settings mentioned in Sec. 3.2 are: (1) class mean in the feature space as the prototype (*relative prototype*); (2) the feature of a certain fixed exemplar as the prototype (*fixed prototype*); (3) the feature of each exemplar as a prototype (*multi-prototype*). From Table 6, the conclusion is that *relative prototype* is the most robust and *fixed prototype* is unstable which conforms to our analysis in Sec 3.2. Although *multi-prototype* gives similar performance to *relative prototype*, the former is more efficient in that it computes the distance between the test sample and each prototype while the latter only needs to calculate one distance. Based on the reasons above, we use *relative prototype* by default.

Prototype-based Classification Strategy for Training. We conduct the following comparison with or without the *selection* functions[2] and the results are shown in Table 7. From that, it can be observed that *selection* functions generally achieve superior performance than the vanilla version. Especially on a more difficult dataset SVHN, the discrepancy in accuracy is much higher. The reason might be that *selection* functions incur useful perturbation when

[2]By "without *selection* functions", we mean that Eq. 7 is used for both training and testing.

Model	Init.	MNIST		F-MNIST		SVHN	
		ACC	FID	ACC	FID	ACC	FID
IntroGAN	Rand. (all)	92.32	146.05	85.81	36.36	51.99	119.73
	Rand. (excl.)	96.01	110.98	87.49	28.93	76.51	**85.90**
	Improved	**97.41**	**8.90**	**88.17**	**25.45**	**77.26**	91.64
MeRGAN	Rand. (all)	61.71	169.04	80.57	38.60	42.61	128.34
	Rand. (excl.)	**97.74**	10.59	77.23	52.76	53.55	**98.03**
	Improved	97.66	**10.23**	**82.93**	**25.56**	**53.70**	99.67
DGR	Rand. (all)	73.73	118.92	69.00	113.32	66.35	109.67
	Rand. (excl.)	74.69	59.13	**70.00**	**88.43**	66.53	116.72
	Improved	**89.88**	**49.16**	68.35	107.84	**69.35**	**108.52**

Table 5: The overall performance of different methods using different initialization strategies on (Fashion-)MNIST and SVHN. 'init' is short for initialization. ACC and FID are short for TA-ACC and TA-FID respectively similar to Table 3 in the main paper. The best score is highlighted in **bold** (for FID, the lower, the better). Init.=Initialization. Excl.=Exclusive.

Cls. strategy	MNIST		Fashion-MNIST		SVHN	
	ACC	FID	ACC	FID	ACC	FID
Fixed prototype	74.40	109.09	62.85	26.71	41.27	102.41
Multi-prototype	97.37	**8.81**	85.88	**24.64**	60.99	108.36
Relative prototype	**97.41**	8.90	**88.17**	25.45	**77.26**	**91.64**

Table 6: The results of IntroGAN using different prototype-based classification strategies on Fashion-MNIST and SVHN. The best score is highlighted in **bold**. Cls.=Classification.

estimating the real class mean and may increase discriminability. We use *max* as the *selection* function by default based on its superiority on SVHN.

6.5. Memory Requirements and Running Time

In Sec. 4.1 of the main paper, it is mentioned that ESGR [18] is not implemented due to its memory inefficiency. In Table 8, we list the memory require-

Settings	**MNIST**		**F-MNIST**		**SVHN**	
	ACC	FID	ACC	FID	ACC	FID
IntroGAN	97.21	17.84	87.41	25.95	70.64	107.09
IntroGAN (mean)	98.00	**8.26**	**89.92**	**25.44**	75.24	107.16
IntroGAN (random)	**98.03**	8.92	88.75	26.72	70.77	105.04
IntroGAN (max)	97.41	8.90	88.17	25.45	**77.26**	**91.64**
IntroGAN (min)	96.37	16.00	88.25	25.59	74.64	106.36

Table 7: The overall performance of IntroGAN with different selection functions (Eq. 9) on different datasets. "IntroGAN" is the vanilla version where Eq. 7 is used for training.

Model	**(Fashion-)MNIST**	**SVHN**	**ImageNet-Dogs**
IntroGAN	9.58MB+0.15MB	9.92MB+0.59MB	121.82MB+2.34MB
MeRGAN	9.73MB	10.07MB	122.29MB
DGR	13.49MB	13.97MB	196.22MB
ESGR	98.28MB	102.88MB	3710.76MB

Table 8: Memory cost of different methods on three datasets. IntroGAN has two kinds of memories: the model (left) and the exemplars (right). Since we use the same network architectures for MNIST and Fashion-MNIST, they have the same memory cost and their results are merged in the single column.

ments of mainly compared methods that have the potential to perform joint incremental generation and classification. From the table, it can be observed that the memory overload of ESGR is much higher than other methods, since it trains a generator for each class. Among other methods, IntroGAN and MeRGAN are almost the same while DGR requires more memory (about 150% of IntroGAN/MeRGAN). The reason is that the classifier and the discriminator are separate in DGR but shared in IntroGAN/MeRGAN.

Although IntroGAN is memory efficient among the GAN-based methods listed above, one may still worry that GAN itself takes too much memory. What if we use the same storage of IntroGAN to store the original images instead of

Model	(Fashion-)MNIST	SVHN	ImageNet-Dogs
IntroGAN	0.037s	0.044s	1.782s
MeRGAN	0.032s	0.040s	1.082s
DGR	0.031s	0.033s	1.006s
ESGR	0.053s	0.057s	9.430s

Table 9: Running time of different methods on three datasets. For convenience, we record the time for one iteration, which is enough to reflect the relative speed of these methods.

GAN model and train a conventional classifier? Such a analysis is elaborated in the supplementary material.

As for the running time, we only estimate the time for one iteration in the first increment for convenience. The reason is that these methods run for the same number of iterations, thus the time for one iteration is enough to reflect the relative speed of these methods. The experiments are performed on one TITAN Xp GPU processor. The statistics of the running time is shown in Table 6.5.

From the results, it can be seen that DGR is the fastest one, because the generator and the classifier are separate. Therefore, the generator loss does not need to back propagate the gradients to update the classifier weights, and vice versa, which saves some time. IntroGAN and MeRGAN need more time than DGR, where IntroGAN is a little more time-consuming. The main reason is that obtaining the prototypes requires an extra forward pass of the network for each iteration. However, this phenomenon can be alleviated by updating the prototypes at a certain interval, especially at the end of training since there is no need to update the prototypes so often (at each iteration).

6.6. Memory Comparisons for GANs and Real Samples

Although IntroGAN is memory efficient among the GAN-based methods listed in Table 8, one may still worry that GAN itself takes too much memory. What if we use the same storage of IntroGAN to store the original images instead of GAN model and train a conventional classifier? However, we should notice that such an approach can no longer perform incremental generation and we can only evaluate its incremental classification performance. The method itself is a special Joint Training approach which only uses a subset of the whole dataset (denoted as *subset*). To calculate how many samples needed in *subset* for fair comparisons with IntroGAN, we show statistics of model sizes and dataset sizes in Table 10. And the number of samples in *subset* is calculated by:

$$\#Subset = \frac{\#Full \times (S_{IntroGAN} - S_{JT})}{S_{Full}} \tag{5}$$

The number of samples in *subset* are shown in the last column of Table 10 (i.e. column #Subset). Note that on ImageNet-Dogs the calculated #Subset has more samples than the original dataset, therefore we use all training samples. For convenience, we assume each class has the same number of samples (i.e. each class has $\frac{\#subset}{\#class}$ samples) and the experimental results are shown in Table 11. From the table we can see that using a subset can have a higher TA-ACC, indicating that using the same memory to store the compressed real images can yield better result in incremental classification than GAN-based methods. This is not surprising because current GANs care more about the image quality instead of the memory efficiency and we did not train as many iterations as ordinary GANs in order to accelerate training in incremental scenarios. The less satisfactory result is common for GAN-based methods, and we believe it can be alleviated by using more lightweight GAN structure [28] or other memory efficiency techniques such as introducing compression of CNNs into GANs, which is beyond the scope of this paper.

7. Class Orders

In Table 2 of the main paper, we mention that different number of class orders are used for (Fashion-)MNIST, SVHN and ImageNet-Dogs. Below are the details of the class orders:

(Fashion-)MNIST/SVHN

Dataset	#Full	S_{Full}	$S_{IntroGAN}$	S_{JT}	#Subset
MNIST	60000	9.45MB	9.73MB	4.07MB	35937
Fashion-MNIST	60000	24.2MB	9.73MB	4.07MB	13476
SVHN	73257	173.61MB	10.51MB	4.08MB	2713
ImageNet-Dogs	35907	39.09MB	124.16MB	75.01MB	45148

Table 10: Statistics of the model and dataset size to calcuate how many samples should be used in the subset (i.e. the #Subset column). #Full is the total number of samples in the full dataset. S_{Full} is the full dataset size. $S_{IntroGAN}$ is the size of the IntroGAN's model plus the stored exemplars. S_{JT} is the size of Joint Training network. #Subset is the calculated numbers of samples in the subset.

Dataset	IntroGAN	Subset
MNIST	97.83	99.12
Fashion-MNIST	87.58	92.47
SVHN	71.03	80.02
ImageNet-Dogs	32.04	34.45

Table 11: TA-ACC of IntroGAN and the conventional way that uses the same memory size to store real images on four datasets (denoted as Subset because it is a subset of the full dataset).

Order 1: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Order 2: 1, 5, 9, 4, 2, 6, 8, 7, 0, 3

Order 3: 7, 2, 4, 3, 9, 5, 6, 8, 1, 0

Order 4: 3, 1, 5, 9, 7, 2, 4, 6, 0, 8

Order 5: 5, 3, 1, 0, 6, 9, 7, 4, 8, 2

The label 0-9 above is based on the definition in [15].

ImageNet-Dogs

Order 1: 172, 98, 86, 68, 115, 197, 42, 88, 199, 97, 160, 131, 31, 179, 41, 17, 206, 180, 104, 76, 188, 19, 203, 62, 30, 65, 20, 24, 44, 169

Order 2: 199, 116, 129, 111, 178, 85, 186, 62, 117, 123, 26, 45, 143, 114, 67, 2, 81, 190, 167, 32, 68, 105, 97, 96, 112, 48, 93, 140, 144, 20

The label above is based on the label definition in [18].

References

[1] T. Lesort, H. Caselles-Dupré, M. Garcia-Ortiz, A. Stoian, D. Filliat, Generative models from the perspective of continual learning, in: IJCNN, 2019, pp. 1–8.

[2] G. M. van de Ven, H. T. Siegelmann, A. S. Tolias, Brain-inspired replay for continual learning with artificial neural networks, Nature Communications 11 (1) (2020) 1–14.

[3] K. Deja, P. Wawrzyński, D. Marczak, W. Masarczyk, T. Trzciński, Binplay: A binary latent autoencoder for generative replay continual learning, in: IJCNN, 2021, pp. 1–8.

[4] J. Ho, A. Jain, P. Abbeel, Denoising diffusion probabilistic models, NeurIPS 33 (2020) 6840–6851.

[5] F.-A. Croitoru, V. Hondru, R. T. Ionescu, M. Shah, Diffusion models in vision: A survey, IEEE Transactions on Pattern Analysis and Machine Intelligence (01) (2023) 1–20.

[6] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training gans, in: NeurIPS, 2016, pp. 2234–2242.

[7] P. Dhariwal, A. Nichol, Diffusion models beat gans on image synthesis, NeurIPS 34 (2021) 8780–8794.

[8] E. Luhman, T. Luhman, Knowledge distillation in iterative generative models for improved sampling speed, arXiv preprint arXiv:2101.02388 (2021).

[9] D. Bau, J.-Y. Zhu, H. Strobelt, B. Zhou, J. B. Tenenbaum, W. T. Freeman, A. Torralba, Gan dissection: Visualizing and understanding generative adversarial networks, in: ICLR, 2018.

[10] R. Gao, W. Liu, Ddgr: continual learning with deep diffusion-based generative replay, in: ICML, 2023, pp. 10744–10763.

[11] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, C. H. Lampert, iCaRL: Incremental classifier and representation learning, in: CVPR, 2017, pp. 2001–2010.

[12] F. M. Castro, M. J. Marín-Jiménez, N. Guil, C. Schmid, K. Alahari, End-to-end incremental learning, in: ECCV, 2018, pp. 233–248.

[13] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[14] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[15] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint arXiv:1708.07747 (2017).

[16] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, Reading digits in natural images with unsupervised feature learning, in: NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.

[17] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. C. Courville, Improved training of wasserstein gans, in: NeurIPS, 2017, pp. 5769–5779.

[18] C. He, R. Wang, S. Shan, X. Chen, Exemplar-supported generative reproduction for class incremental learning, in: BMVC, 2018, pp. 3–6.

[19] A. Khosla, N. Jayadevaprakash, B. Yao, F.-F. Li, Novel dataset for fine-grained image categorization: Stanford dogs, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop on Fine-Grained Visual Categorization (FGVC), Vol. 2, 2011, p. 1.

[20] T. Miyato, T. Kataoka, M. Koyama, Y. Yoshida, Spectral normalization for generative adversarial networks, in: ICLR, 2018.

[21] A. Odena, C. Olah, J. Shlens, Conditional image synthesis with auxiliary classifier gans, arXiv preprint arXiv:1610.09585 (2016).

[22] C. Wu, L. Herranz, X. Liu, J. van de Weijer, B. Raducanu, et al., Memory replay GANs: Learning to generate new categories without forgetting, in: NeurIPS, 2018, pp. 5962–5972.

[23] H. Shin, J. K. Lee, J. Kim, J. Kim, Continual learning with deep generative replay, in: NeurIPS, 2017, pp. 2994–3003.

[24] A. Chaudhry, P. K. Dokania, T. Ajanthan, P. H. Torr, Riemannian walk for incremental learning: Understanding forgetting and intransigence, in: ECCV, 2018, pp. 532–547.

[25] Z. Li, D. Hoiem, Learning without forgetting, in: ECCV, 2016, pp. 614–629.

[26] O. Ostapenko, M. Puscas, T. Klein, P. Jahnichen, M. Nabi, Learning to remember: A synaptic plasticity driven framework for continual learning, in: CVPR, 2019, pp. 11321–11329.

[27] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, Z. Zhang, Y. Fu, Incremental classifier learning with generative adversarial networks, arXiv preprint arXiv:1802.00853 (2018).

[28] T. Shimizu, J. Xu, K. Tasaka, MobileGAN: Compact network architecture for generative adversarial network, in: Asian Conference on Pattern Recognition, Springer, 2019, pp. 326–338.