In this document, we provide additional materials on the proposed method for few-shot dataset distillation (DD) that cannot be accommodated in the main paper due to the page limitation. We first provide detailed algorithms for both our method and baselines for comparison. Then, more quantitative and qualitative results are included.

A. More Implementing Details

A.1. Our Method

Pre-training. Here, we provide detailed algorithmic implementation to Alg. 2 of the main paper, the translative pre-training algorithm for distillation space. The key design of the translative pre-training pipeline lies in the separation of two stages: dataset distillation in an arbitrary but fixed neural network and the translation to the desired space. In practice, instead of the version shown in the main paper is shown in Fig. 1. It adopts an auto-encoder structure, the pre-trained translator can also be adapted to datasets with different resolutions beyond the one used during pre-training. Please refer to the following section for details.

Algorithm 1 Caching Stage.

Input: \( \mathcal{Z} \): A Large Dataset; \( \theta \): An Arbitrary Random Neural Network; \( T \): Number of Update Steps for Synthetic Data; \( \alpha \): Learning Rate for Synthetic Data; \( N \): Number of Cached Subsets; \( C_{\text{max}} \) and \( C_{\text{min}} \): Maximal and Minimal Number of Classes in a Subset; \( M \): Maximal Number of Images in a Subset; \( I_{\text{max}} \): Maximal Number of Images in a Synthetic Dataset.

Output: \( \mathcal{D} \): A Dataset Used for Pre-training.

1. \( \mathcal{D} \leftarrow \emptyset \);
2. for \( N \) subsets in parallel do
   ▷ Data preparation.
   3. Select an integer \( C \) randomly from \([C_{\text{min}}, C_{\text{max}}]\);
   4. Select \( C \) classes randomly from \( \mathcal{Z} \);
   5. Select an integer \( I \) randomly from \([1, I_{\text{max}}]\);
   6. \( X_T, Y_T, X_S, Y_S = [], [], [], [] \);
   7. for \( 1 \leq c \leq C \) do
      8. \( X_{T,c} \leftarrow \lfloor \frac{|\mathcal{X}_c|}{C} \rfloor \) random images in class \( c \);
      9. \( Y_{T,c} \leftarrow \text{OneHot}([c] \times \lfloor \frac{M}{C} \rfloor) \);
     10. \( X_{S,c} \leftarrow X_{T,c} \); \( Y_{S,c} \leftarrow Y_{T,c} \);
     11. \( X_T \leftarrow [X_T; X_{T,c}]; Y_T \leftarrow [Y_T; Y_{T,c}] \);
     12. \( X_S \leftarrow [X_S; X_{S,c}]; Y_S \leftarrow [Y_S; Y_{S,c}] \);
   end for
   ▷ Dataset distillation in one network.
   13. for \( T \) steps do
      14. \( w_{S,0} \leftarrow f_\theta(X_S)^T(f_\theta(X_S)f_\theta(X_S)^T)^{-1}Y_S \);
      15. \( \mathcal{L} = \| f_\theta(DSA(X_T))w_{S,0} - Y_T \|^2 \);
      16. \( X_S \leftarrow X_S - \alpha \nabla_{X_S} \mathcal{L} \);
      17. \( Y_S \leftarrow Y_S - \alpha \nabla_{Y_S} \mathcal{L} \);
   end for
   18. \( \mathcal{D} \leftarrow \mathcal{D} \cup \{(Idx(X_T), X_S, Y_S)\} \);
end for

Adaptation. Given a pre-trained translator, it requires a small number of adaptation steps in general for the target dataset. The detailed adaptation algorithm is shown in Alg. 3. The hyper-parameters are summarized in Tab. 1 of the main paper.

Translator. The detailed architecture of the translator in this paper is shown in Fig. 1. It adopts an auto-encoder structure. To better inherit the useful information in the input, the network learns the difference between desired output and input. Notably, due to the fully-convolutional structure, the pre-trained translator can also be adapted to datasets with different resolutions beyond the one used dur-

A.2. Baselines

Benchmark. The baseline algorithm, used in Fig. 1 and Tab. 2 in the main paper, and Tab. 1 here, is shown in Alg. 4.
Algorithm 2 Pre-training Stage.
\textbf{Input:} \(Z\): A Large Dataset; \(\Theta\): Distribution for Initializing Neural Networks; \(\eta\): Learning Rate for Translator; \(D\): Dataset Cached by Alg. 1; \(B\): Batch Size.
\textbf{Output:} \(\omega\): A Pre-trained Translator.
1. Initialize \(\omega\) randomly;
2. repeat
3. for \(B\) samples in parallel do
4. \((I dx, X_S^t, Y_S) \leftarrow \) a random sample from \(D\);
5. Fetch \(X_T, Y_T\) from \(Z\) with \(I dx\);
6. \(X_S \leftarrow G_\omega(X_S^t)\);
7. Randomly sample \(\theta^t \sim \Theta\);
8. \(w_{S, \theta^t}^* \leftarrow f_{\theta^t}(X_S)^T (f_{\theta^t}(X_S) f_{\theta^t}(X_S)^T)^{-1} Y_S\);
9. \(\nabla^\omega \leftarrow \nabla_{\omega} \|f_{\theta^t}(X_T) w_{S, \theta^t}^* - Y_T\|^2\);
10. end for
11. \(\omega \leftarrow \omega - \eta \frac{1}{B} \sum_{i=1}^{B} \nabla^\omega\);
12. until Converge

\begin{figure}[h]
\centering
\includegraphics[width=0.7\textwidth]{architecture.png}
\caption{Architecture of our translator.}
\end{figure}

In experiments, we maintain the same total number of update steps for synthetic data. When the number of networks \(R\) is 1, it corresponds to the setting of 1 Net in Tab. 1. And when \(R = T\), it is the setting of Baseline.

Relationships with state of the arts. The algorithm shown in Alg. 4 is based on neural feature regression (NFR) [6]. It first uses a neural network as a feature extractor to map the original input to a feature space and then performs kernel ridge regression. On the one hand, it behaves essentially the same as RFAD [5] and the only difference is that it uses a new random neural network in each step of updating synthetic data. On the other hand, compared with FRePo, it does not update networks via current synthetic data or store the networks in a pool. That is why FRePo can further improve performance in many cases. However, these related works have no constraint on the total number of networks, which negatively affects the efficiency of dataset distillation, as illustrated in Fig. 1 of the main paper. We are thus motivated by this observation and dedicated to studying the problem of few-shot dataset distillation which only adopts a limited number of networks.

Algorithm 3 Adaptation Stage.
\textbf{Input:} \((X_T, Y_T)\): A Target Dataset to be Distilled; \(\Theta\): Distribution for Initializing Neural Networks; \(\theta\): The Random Neural Network Used in the Cache Stage; \(\eta\): Learning Rate for Translator; \(I\): A List of Numbers of Images per Class for Adaptation; \(T\): Number of Update Steps for Synthetic Data; \(\alpha\): Learning Rate for Synthetic Data; \(\omega_0\): A Pre-trained Translator; \(S\): Number of Adaptation Steps for Translator; \(B\): Batch Size.
\textbf{Output:} \(\omega\): An Adapted Translator.
1. \(D \leftarrow \emptyset\);
2. for \(I\) in \(I\) in parallel do
3. \(\triangleright\) Synthetic data initialization.
4. \(X_S, Y_S = [], []\);
5. for \(1 \leq c \leq C\) do
6. \(X_{S,c} \leftarrow I\) random images in class \(c\);
7. \(Y_{S,c} \leftarrow \text{OneHot}(c) \times I\);
8. \(X_S \leftarrow [X_S; X_{S,c}], Y_S \leftarrow [Y_S; Y_{S,c}]\);
9. end for
\(\triangleright\) Dataset distillation in one network.
10. \(X_T^\theta \leftarrow f_\theta(X_T)\);
11. for \(T\) steps do
12. \(w_{S, \theta^t}^* \leftarrow f_{\theta^t}(X_S)^T (f_{\theta^t}(X_S) f_{\theta^t}(X_S)^T)^{-1} Y_S\);
13. \(X_S \leftarrow X_S - \alpha \nabla_{X_S} \|X_T^\theta w_{S, \theta^t}^* - Y_T\|^2\);
14. \(Y_S \leftarrow Y_S - \alpha \nabla_{Y_S} \|X_T^\theta w_{S, \theta^t}^* - Y_T\|^2\);
15. end for
16. \(D \leftarrow D \cup \{(X_S, Y_S)\}\);
17. end for
\(\triangleright\) Adapt the translator to the desired space.
18. \(\omega \leftarrow \omega_0\);
19. for \(S\) steps do
20. \(\triangleright\) for \(B\) samples in parallel do
21. \((X_S^t, Y_S^t) \leftarrow \) a random sample from \(D\);
22. \(X_S \leftarrow G_\omega(X_S^t)\);
23. Randomly sample \(\theta^t \sim \Theta\);
24. \(w_{S, \theta^t}^* \leftarrow f_{\theta^t}(X_S)^T (f_{\theta^t}(X_S) f_{\theta^t}(X_S)^T)^{-1} Y_S\);
25. \(\nabla^\omega \leftarrow \nabla_{\omega} \|f_{\theta^t}(X_T) w_{S, \theta^t}^* - Y_T\|^2\);
26. end for
27. end for

B. More Experimental Results

Results on larger datasets. Although trained with only 100 classes and \(32 \times 32\) resolution at most during pre-training, we demonstrate that it is also feasible for the pre-trained translator to be adapted to target datasets with more classes and higher resolutions. Following the settings in the main paper, Tab. 1 shows experimental results on ImageNet1k [2] with 1k classes under \(32 \times 32\) resolution and ImageNette [3] with 10 classes under \(128 \times 128\) resolution. For ImageNet1k, although the performance is bottle-necked...
by the challenge of scalability in DD [1], the improvement over baseline methods is consistent and it achieves \(\sim 2\times\) acceleration compared with Baseline. For ImageNette, we first conducted distillation under the \(32 \times 32\) resolution in the pre-defined network and then up-sample the distilled results to \(128 \times 128\) resolution with bi-linear interpolation as input to the translator. Due to the discrepancy in resolutions, the performance of results after translation is worse than that of 1 Net in this case, which is distilled in a network directly under \(128 \times 128\) resolution. However, after a small number of adaptation steps, with \(3 \sim 4\times\) acceleration, it can even surpass Baseline which is distilled by enormous networks under the large resolution.

More results on cross-IPC generalization. Here, we provide more results on cross-IPC generalization for an adapted translator on some seen IPCs. As shown in the red curve of Fig. 2(left), on CIFAR10 [4], we adapt the translator on 1 and 50 IPCs and report the performance for a variety of IPCs including both seen and unseen ones. As references, we also report the results without adapting the translator and adapting the translator on the specific IPC only in the green and yellow curves respectively. We can find that the performance on unseen IPCs is close to or sometimes even slightly better than that by adapting specifically. Similarly, in Fig. 2(right), we conduct experiments in the same way on ImageNette. For the red curve, the translator is adapted on 1 and 12 IPCs. We observe that the generalization is satisfactory and can approach the yellow curve when IPC is large. In all cases, it works significantly better than no adaptation, which indicates that the adaptation stage helps the translator encode useful knowledge of the whole target dataset, not limited to the knowledge required by the seen IPCs.

More qualitative visualization. We provide more qualitative visualizations on CIFAR10 and ImageNette datasets in Fig. 3 and Fig. 4 respectively, including results distilled by the pre-defined network, results produced by the pre-trained translator without adaptation, and results produced by the translator after adaptation as seen and unseen IPCs.
The observations are consistent with those in the main paper: the pre-trained translator mainly modifies global styles and colors while the adapted translator adds more local details, and the learned textures in the translator are also transferable to unseen IPCs.

References


Figure 3. Qualitative analysis of results produced by our method for IPC 10 on CIFAR10.
Figure 4. Qualitative analysis of results produced by our method for IPC 10 on ImageNette.