Beyond Soft Label: Dataset Distillation via Orthogonal Gradient Matching

Supplementary Material

8. Dive into Optimizers

Adam & AdamW. Both Adam [19] and AdamW are optimizers tailored for vectorized gradients. Their difference lies in the position of weight decay, which will not be distinguished below. Compared to SGD, Adam uses the second-order moment of the gradient, *i.e.*, $g * g$, to approximate the diagonal of the second-order gradient, namely the Hessian matrix, thereby accelerating the training of neural networks. The pseudo-algorithm of Adam is shown in Algorithm 3. By switching off exponential moving averages, Adam is equal to sign gradient descent [1] and its update rule is shown in Algorithm 4, which is the same as Equation 4.

Algorithm 3 Adam

Initialize $w_0, \eta, m_0, v_0, \hat{m}_0, \hat{v}_0, \beta_1, \beta_2$
for $t = 1$ to T **do**
  $g_t \leftarrow \nabla_w \mathcal{L}(w_{t-1})$ ▷ Gradient
  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$ ▷ 1-st Moment
  $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$ ▷ 2-nd Moment
  $\hat{m}_t \leftarrow m_t / (1-\beta_1^t)$
  $\hat{v}_t \leftarrow v_t / (1-\beta_2^t)$
  $w_t \leftarrow w_{t-1} - \eta \hat{m}_t / \sqrt{\hat{v}_t + \epsilon}$ ▷ Update
end for

Algorithm 4 Adam w/o EMA

Initialize $w_0, \eta, m_0, v_0, \hat{m}_0, \hat{v}_0$
for $t = 1$ to T **do**
  $G_t \leftarrow \nabla_W \mathcal{L}(W_{t-1})$ ▷ Gradient
  $m_t \leftarrow g_t$
  $v_t \leftarrow g_t^2$
  $\hat{m}_t \leftarrow m_t$
  $\hat{v}_t \leftarrow v_t$
  $w_t \leftarrow w_{t-1} - \eta \text{sign}(g_t)$ ▷ Update
end for

Shampoo & Muon. Different from Adam, which only approximates the diagonal of the Hessian matrix, Shampoo [13] and Muon [17] directly leverage the covariance of the gradient, *i.e.*, $GG^\top$, to introduce the information of the Hessian matrix and update parameters, which are more effective for matrix-wise gradient. Here, we take Muon as an example, which orthogonalizes the scale information of gradients by setting all singular values to one. The update rule of Muon is defined as

$$W_{t+1} = W_t - \eta \cdot U_t V_t^\top, \quad \text{SVD}(G_t) = U_t S_t V_t^\top, \quad (19)$$

where G_t is the matrix-wise gradient in the t-th iteration. Notably, the singular vectors $U_t V_t^\top$ can be obtained by

$$U_t V_t^\top = U_t \frac{1}{\sqrt{S_t^2}} U_t^\top U_t S_t V_t^\top = (G_t G_t^\top)^{-\frac{1}{2}} G_t, \quad (20)$$

where $(G_t G_t^\top)^{-\frac{1}{2}}$ is the pseudo-inverse of G_t, containing the information of the Hessian matrix. In practice, calculating the pseudo-inverse or SVD is time-consuming. Therefore, Muon adopts the Newton-Schulz matrix iteration to approximate the matrix orthogonalization process, which only slightly increases the time complexity. We omit it for clarity.

The above comparison between Adam and Muon reveals that the second-order gradient, *i.e.*, Hessian matrix, is crucial for model optimization, which sheds light on the further direction of DD.

9. Details of Experimental Investigation

In this section, we introduce the experimental details in Section 3.2.

Dataset. The experiments are conducted on the full ImageNet-1K dataset and its subset, which contains 1,281,167 and 10,000 images, respectively. The subset of ImageNet has ten images per class (IPC=10), which are randomly selected from the original dataset. The subset data is provided by Xiao et al. [38] and can be downloaded from Huggingface[2]. The evaluation is conducted on the validation dataset of ImageNet-1K, which contains 50,000 images in total.

Optimizer. This experiment aims to verify the effectiveness of two optimizers: SGD and Muon [17].

For the gradients with different shapes, SGD can optimize them with the same rule, *i.e.*, $w_{t+1} = w_t - \eta g_t$ and $W_{t+1} = W_t - \eta G_t$, where g_t and G_t denotes the vector-wise and matrix-wise gradient in the t-th iteration, respectively.

In contrast, Muon needs to apply SVD on the gradients to eliminate their scales, which is infeasible for vector-wise gradients. To address this issue, Muon adopts a mix-optimization strategy, which uses SGD to optimize vector-wise gradients and only applies SVD on matrix-wise gradients. The pseudo-algorithm can be seen in Algorithm 5.

The configurations of the two optimizers are listed in Table 7 below.

[2] https://huggingface.co/datasets/he-yang/2025-rethinkdc-imagenet-random-ipc-10

Algorithm 5 Mix-optimization of Muon

Initialize W_0, η
for $t = 1$ to T **do**
  $G_t \leftarrow \nabla_W \mathcal{L}(W_{t-1})$ ▷ Gradient
  if G_t.shape > 1 **then** ▷ Matrix-wise
    $G_t \leftarrow G_t$.view(G_t.shape[0], -1)
    $U_t S_t V_t^\top \leftarrow G_t$ ▷ SVD
    $G_t \leftarrow U_t V_t^\top$
  end if
  $W_t \leftarrow W_{t-1} - \eta G_t$ ▷ Update
end for

Table 7. Configurations of SGD and Muon

	SGD	Muon
Epoch (Full / Subset)	60 / 300	60 / 300
Batch Size	256	256
Learning Rate	0.2	0.2
Scheduler	StepLR	Cosine
Momentum	0.9	0.7
Weight Decay	1e-4	1e-4

Training and Evaluation. We evaluate the performance of ResNet-18 after each epoch of training, and report the performance in Figures 4 and 5.

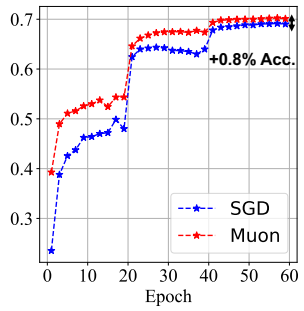


Figure 4. Full ImageNet

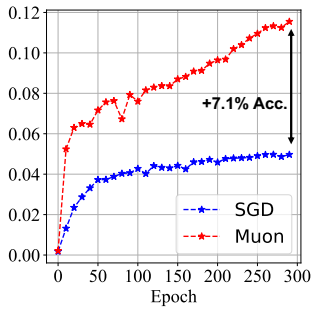


Figure 5. Subset, IPC=10

10. Details of Experiments

Hyperparameters. The hyperparameters used in the training and evaluation are shown in Table 8.

Environment. All experiments are conducted on 4×GeForce RTX 4090.

11. Configuration of OGM

OGM is initialized with RDED, where each synthetic image contains four patches. To prevent the interference between patches, in each iteration, we randomly select a patch and apply the *RandomResizedCrop* on it. The PyTorch-style pseudo-algorithm is shown in Algorithm 6

Table 8. Hyperparameters in training and evaluation.

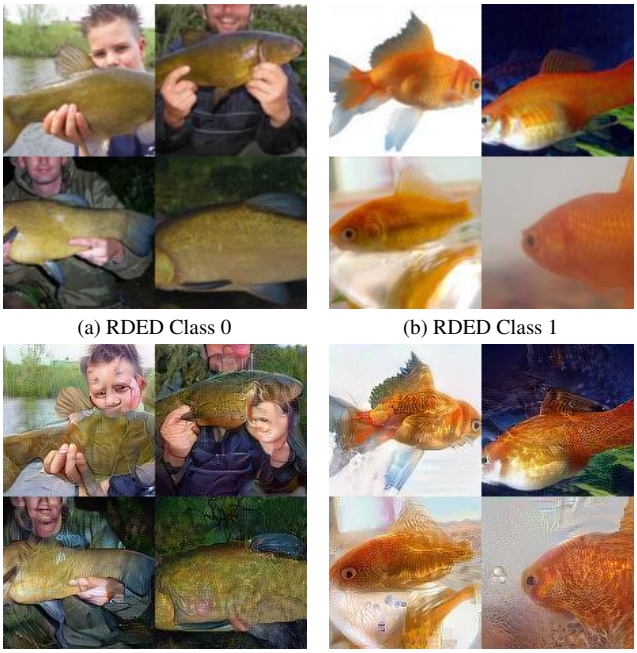
	Image Synthesis	Basic Evaluation (Tables 1 & 2)	Ablation Evaluation (Table 3)
Optimizer	Adam	AdamW	Muon
Learning Rate	0.05	1e-3	0.1
Scheduler	Cosine	Cosine	Cosine
Weight Decay	-	1e-2	1e-4
Momentum	(β_1, β_2)=(0.5, 0.9)	(β_1, β_2)=(0.9, 0.999)	0.7 / 0.9
Epoch	1000	300	300
Batch Size	100	128	128
Augmentation	RandomResizedCrop(0.5) HorizontalFlip	RandomResizedCrop(0.08) HorizontalFlip	RandomResizedCrop(0.08) HorizontalFlip
Loss Function	Equation 11	Cross-entropy (Hard) KL Divergence (Soft)	Cross-entropy (Hard) KL Divergence (Soft)

Algorithm 6 PyTorch code of Patch Extract

```
class PatchExtract(torch.nn.Module):
    def __init__(self, factor):
        super().__init__()
        self.factor = factor

    def forward(self, img):
        B, C, H, W = img.shape
        ph, pw = H // self.factor, W // self.factor

        # Randomly select the index of row and column
        rows = torch.randint(0, self.factor, size=(B,),
            device=img.device)
        cols = torch.randint(0, self.factor, size=(B,),
            device=img.device)

        patches = img.unfold(2, ph, ph).unfold(3, pw, pw)
        batch_idx = torch.arange(B, device=img.device)
        selected = patches[batch_idx, :, rows, cols]

        return selected

# Augmentation
aug_function = transforms.Compose(
    [
        PatchExtract(2),
        RandomResizedCrop(im_size[0], scale=(0.5, 1),
            antialias=True),
        RandomHorizontalFlip()
    ]
)
```

12. Visualization

Finally, we provide a fine-grained comparison between RDED and OGM in Figure 6. It can be observed that the synthetic images of RDED are smoother than OGM, while OGM adds some fine-grained textures to the original patches. We speculate that these textures contribute to the optimization of models. Moreover, this comparison also reflects that OGM captures the knowledge of real datasets rather than relying on the initialization of RDED.



(a) RDED Class 0 (b) RDED Class 1

(c) OGM Class 0 (d) OGM Class 1

Figure 6. Comparison between RDED and OGM