Dataset Condensation/Distillation

Objective: learn a smaller set of synthetic data that is able to train a model to comparable performance as a model trained on a much larger dataset (condense a large dataset into a smaller set of synthetic data)

Existing research

- Dataset Distillation (Wang)
- Flexible Dataset Distillation: Learn Labels Instead of Images (Bohdal)
- Soft Label Dataset Distillation and Text Dataset Distillation (Sucholutsky)
- Generative Teaching Networks: Accelerating Neural Architecture Search by Learning to Generate Synthetic Training Data (Such)
- Mnemonics Training Multi-Class Incremental Learning without Forgetting (Liu)
- Dataset Meta-Learning from Kernel Ridge-Regression (Nguyen)
- Dataset Condensation with Gradient Matching (Bo Zhao)
- Dataset Condensation with Differentiable Siamese Augmentation (Bo Zhao)

	Image	Label	Ideas/Concepts discussed
Dataset Distillation (Wang)	✔		Malicious data poisoning Network Initialization Adapting pre-trained models to new datasets
Flexible Dataset Distillation: Learn Labels Instead of Images (Bohdal)		✔	Detection of Overfitting Ridge Regression, Pseudo-gradients Cross-Dataset distillation Flexibility of Distilled Datasets (number of steps used, changes in optimisation parameters) Cross-Architecture Generalization
Soft Label Dataset Distillation and Text Dataset Distillation (Sucholutsky)	✔	✔ Soft Label	Network Initialization Text Distillation K-Nearest Neighbours Soft Labels
Generative Teaching Networks: Accelerating Neural Architecture Search by Learning to Generate Synthetic Training Data (Such)	✔ OR ✔	✔	Curriculum Weight Normalization Neural Architecture Search

	Image	Label	Ideas discussed
Mnemonics Training Multi-Class Incremental Learning without Forgetting (Liu)	✔		Continual Learning Bi-Level Optimization Program (BOP)
Dataset Meta-Learning from Kernel Ridge-Regression (Nguyen)	✔ OR	✔	ε-approximation (new concept) Privacy-preserving Kernel ridge-regression Robustness to hyperparameters Inducing Points
Dataset Condensation with Gradient Matching (Bo Zhao)	✔		Continual Learning Cross-Architecture Generalization Neural Architecture Search Activation, normalization & pooling
Dataset Condensation with Differentiable Siamese Augmentation (Bo Zhao)	✔		Continual Learning Cross-Architecture Generalization Neural Architecture Search Data augmentation

	Discussion of robustness to choice of initialization (parameters/images)	Cross-Architecture Generalization (application to NAS)	Continual Learning
Dataset Distillation (Wang)	✔		
Flexible Dataset Distillation: Learn Labels Instead of Images (Bohdal)		✔	
Soft Label Dataset Distillation and Text Dataset Distillation (Sucholutsky)	✔		
Generative Teaching Networks: Accelerating Neural Architecture Search by Learning to Generate Synthetic Training Data (Such)	✔	✔	
Mnemonics Training Multi-Class Incremental Learning without Forgetting (Liu)			✔
Dataset Meta-Learning from Kernel Ridge-Regression (Nguyen)	✔	✔	
Dataset Condensation with Gradient Matching (Bo Zhao)		✔	✔
Dataset Condensation with Differentiable Siamese Augmentation (Bo Zhao)	✔	✔	✔

Current state of literature (brief)

Areas that possibly need further investigation/improvement:

- Initialization:
  - Initialization of images
  - Initialization of network parameters/weights (random vs fixed initialization, the distribution that random initialization comes from)
- Cross-Architecture Generalization
  - Some methods are less flexible than others in terms of being over-fitted to training conditions under which it was generated, must used a highly-customised learner (specific image visitation sequence, meta-learned learning rates, number of learning steps etc)
- Performance gap between models trained on small synthetic set and those trained on whole training set (for DD and DC on CIFAR-10)
- All methods so far have only focused on image data, with the exception of TDD focusing on text data. Methods for other types of data have yet been developed (or current methods have not been shown to work with other types of data yet)
- Yet to investigate the effect of dataset distillation/condensation on bias in datasets

Other ideas

- Link with dimensionality reduction techniques (matrix factorization extracts key features of an image)
- Link with LUPI (Learning using privileged information)
- Soft-labels: Investigate whether it is better to separate similar classes (e.g. 3 and 8 in MNIST) to increase the network's ability to discern between them, or keep them together to allow soft-label information to be shared
- KIP: Investigate why sometimes higher corruption in images leads to better test performance

Dataset Distillation

Method: Learn a small number of synthetic data samples not only capturing much of the original training data but also tailored explicitly for fast model training in only a few gradient steps

- Derive the network weights as a differentiable function of synthetic training data. Given this connection, instead of optimizing the network weights for a particular training objective, optimize the pixel values of distilled images.
- Optimise both synthetic data x* and learning rate η*

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \mathcal{L}(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_0) = \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \ell(\mathbf{x}, \theta_1) = \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \ell(\mathbf{x}, \theta_0 - \tilde{\eta} \nabla_{\theta_0} \ell(\tilde{\mathbf{x}}, \theta_0)),$$

Generalizing this to a distribution of random initialized weights:

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \mathbb{E}_{\theta_0 \sim p(\theta_0)} \mathcal{L}(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_0),$$

In practice, distilled data generalizes well to unseen initializations. In addition, these distilled images often look quite informative, encoding the discriminative features of each category

Algorithm 1 Dataset Distillation

Input: $p(\theta_0)$: distribution of initial weights; M: the number of distilled data
Input: α: step size; n: batch size; T: the number of optimization iterations; $\tilde{\eta}_0$: initial value for $\tilde{\eta}$

Initialize $\tilde{\mathbf{x}} = \{\tilde{x}_i\}_{i=1}^{M}$ randomly, $\tilde{\eta} \leftarrow \tilde{\eta}_0$
for each training step $t = 1$ to T **do**
    Get a minibatch of real training data $\mathbf{x}_t = \{x_{t,j}\}_{j=1}^{n}$
    Sample a batch of initial weights $\theta_0^{(j)} \sim p(\theta_0)$
    for each sampled $\theta_0^{(j)}$ **do**
        Compute updated parameter with GD: $\theta_1^{(j)} = \theta_0^{(j)} - \tilde{\eta}\nabla_{\theta_0^{(j)}}\ell(\tilde{\mathbf{x}}, \theta_0^{(j)})$
        Evaluate the objective function on real training data: $\mathcal{L}^{(j)} = \ell(\mathbf{x}_t, \theta_1^{(j)})$
    end for
    Update $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} - \alpha\nabla_{\tilde{\mathbf{x}}}\sum_j \mathcal{L}^{(j)}$, and $\tilde{\eta} \leftarrow \tilde{\eta} - \alpha\nabla_{\tilde{\eta}}\sum_j \mathcal{L}^{(j)}$
end for

Output: distilled data $\tilde{\mathbf{x}}$ and optimized learning rate $\tilde{\eta}$

Extending to multiple GD steps

- analysis of the lower bound of distilled data size shows that any small number of distilled data fail to generalize to arbitrary initial θ_0 , intuitively expected as the optimization target depends on the local behaviour of l around θ_0, which can be drastically different across various initializations
- Extend algorithm 1 such that line 6 updates θ_{i+1} and line 9 backpropagates through all steps
- However, this is memory and computationally expensive, to solve this problem, use back-gradient optimization —> formulates necessary second order terms into efficient Hessia—vector products which can be easily calculated with modern automatic differentiation systems such as PyTorch

Extending to multiple epochs:

- for each epoch, method cycles through all GD steps, where each step is associated with a batch of distilled data
- do not tie the trained learning rates across epochs as later epochs often use smaller learning rates

Distillation for malicious data poisoning

- When a single GD step is applied with the synthetic adversarial data, a well-behaved image classifier catastrophically forgets one category but still maintains high accuracy on other categories

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \mathbb{E}_{\theta_0 \sim p(\theta_0)} \mathcal{L}_{K \to T}(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_0),$$

- K = attacked category, T = target category. L is a classification loss that encourages the network to misclassify images of category K as category T while correctly predicting other images
- Distilled images do not require access to the exact model weights and thus can generalize to unseen models
- Compared to previous data poisoning attack approaches, this approach does not require poisoned training data to be stored and trained on repeatedly, instead attacking the model training in one iteration with only a few data

Experiments:

- Initializations
  - Random initialization: distribution over random initial weights e.g. He Initialization, Xavier Initialization
  - Fixed initialization: a particular fixed network initialized by the method above
  - Random pre-trained weights: distribution over models pre-trained on other tasks or datasets
  - Fixed pre-trained weights: particular fixed network pre-trained on other tasks and datasets
  - Method characterizes domain mismatch via distilled data - Experiments show that a small number of distilled images are sufficient to quickly adapt CNN models to new datasets and tasks
- Baselines: Random real images, optimized real images, k-means, average real images
- Datasets: MNIST, CIFAR10

Limitations of method:

Method is sensitive to distribution of initializations —> want to investigate other initialization strategies with which dataset distillation can work well

Dataset Condensation with Gradient Matching

Method: Learn a synthetic set S with fixed class labels that

- achieves comparable generalization performance to training on the original set T
- converges to a similar solution for the parameters of the neural network
- Works with a distribution of random initializations of parameters

$$\min_{\mathcal{S}} \mathrm{E}_{\boldsymbol{\theta}_0 \sim P_{\boldsymbol{\theta}_0}} [D(\boldsymbol{\theta}^{\mathcal{S}}(\boldsymbol{\theta}_0), \boldsymbol{\theta}^{\mathcal{T}}(\boldsymbol{\theta}_0))] \quad \text{subject to} \quad \boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}(\boldsymbol{\theta}_0))$$
$$\text{where } \boldsymbol{\theta}^{\mathcal{T}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}(\boldsymbol{\theta}_0)).$$

θ^S = parameters of network trained on synthetic set S
θ^T = parameters of network trained on normal set T
θ_0 = initialization of network parameters

$P_{\theta 0}$ = initialization of network parameters
$D(\cdot\ ,\ \cdot)$ = distance function
L^S = loss function

Theory:

Use back-optimization approach that redefines θ^S as the output of an incomplete optimisation:

$$\boldsymbol{\theta}^{\mathcal{S}}(\mathcal{S}) = \texttt{opt-alg}_{\boldsymbol{\theta}}(\mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}), \varsigma)$$

In practice, θ^T can be trained first in an offline stage and then used as the target parameter vector in training θ^S. However, there are **two challenges**:

1. The distance between θ^T and intermediate values of θ^S can be too big in the parameter space with multiple local minima traps along the path → optimal solution may be too difficult to reach
2. opt-alg involves a limited number of steps as a trade-off between speed and accuracy → steps may be insufficient to reach optimal solution

To address the challenges, the authors proposed the **curriculum-based gradient-matching approach.**

Theory:

- Key idea: θ^S should be not only close to the final θ^T, but should also follow a similar path as θ^T
- $D(\theta^S, \theta^T) \approx 0$ at each iteration
- Using gradient descent as optimisation algorithm, the problem becomes:

$$\min_{\mathcal{S}} \mathrm{E}_{\boldsymbol{\theta}_0 \sim P_{\boldsymbol{\theta}_0}} [\sum_{t=0}^{T-1} D(\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_t), \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}(\boldsymbol{\theta}_t))].$$

$$D(\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{S}}, \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{T}}) = \sum_{l=1}^{L} d(\nabla_{\boldsymbol{\theta}^{(l)}} \mathcal{L}^{\mathcal{S}}, \nabla_{\boldsymbol{\theta}^{(l)}} \mathcal{L}^{\mathcal{T}})$$

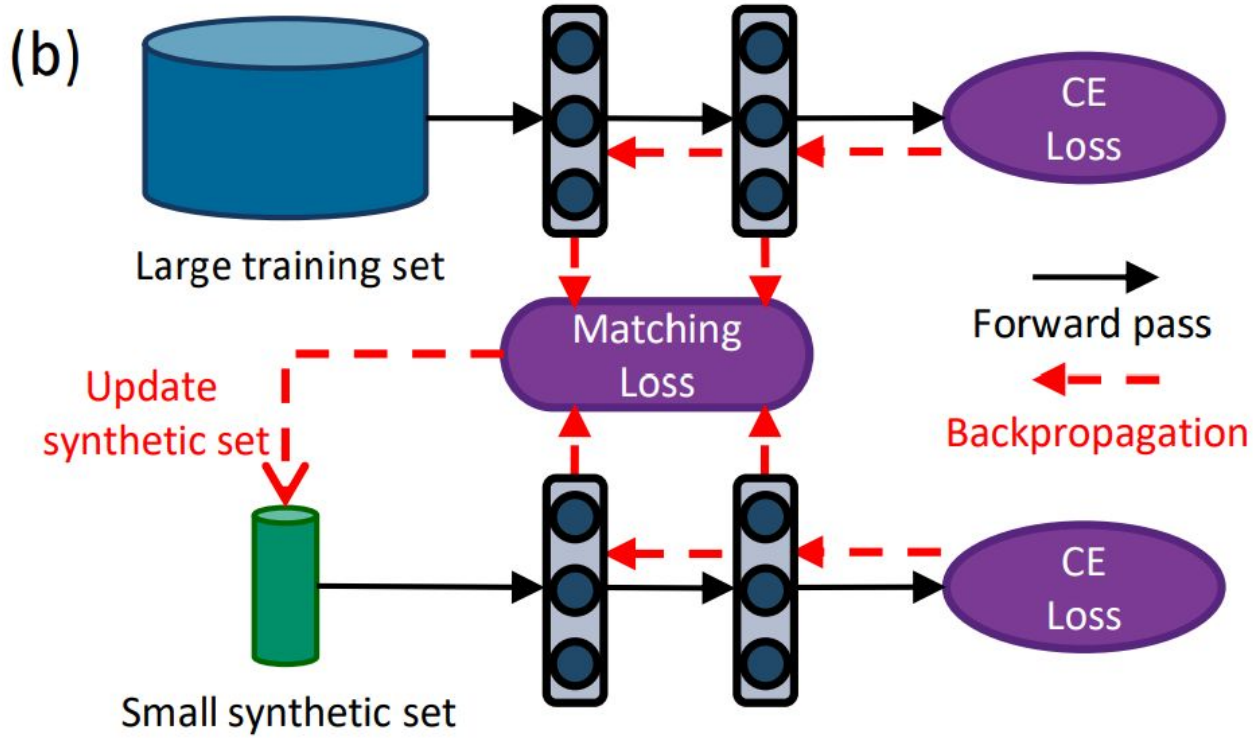
L = number of layers with weights

$$d(\mathbf{A}, \mathbf{B}) = \sum_{i=1}^{\text{out}} \left(1 - \frac{\mathbf{A_{i\cdot}} \cdot \mathbf{B_{i\cdot}}}{\|\mathbf{A_{i\cdot}}\| \|\mathbf{B_{i\cdot}}\|}\right)$$

out = number of output channels

Objective: match the gradients for the real and synthetic training loss w.r.t. θ via updating the synthetic set S

Flowchart of gradient-matching method



Algorithm 1: Dataset condensation with gradient matching

Input: Training set $\mathcal{T}$

Required: Randomly initialized set of synthetic samples $\mathcal{S}$ for C classes, probability distribution over randomly initialized weights $P_{\boldsymbol{\theta}_0}$, deep neural network $\phi_{\boldsymbol{\theta}}$, number of outer-loop steps K, number of inner-loop steps T, number of steps for updating weights $\varsigma_{\boldsymbol{\theta}}$ and synthetic samples $\varsigma_{\mathcal{S}}$ in each inner-loop step respectively, learning rates for updating weights $\eta_{\boldsymbol{\theta}}$ and synthetic samples $\eta_{\mathcal{S}}$.

for $k = 0, \cdots, K-1$ **do**

  Initialize $\boldsymbol{\theta}_0 \sim P_{\boldsymbol{\theta}_0}$

  for $t = 0, \cdots, T-1$ **do**

    for $c = 0, \cdots, C-1$ **do**

      Sample a minibatch pair $B_c^{\mathcal{T}} \sim \mathcal{T}$ and $B_c^{\mathcal{S}} \sim \mathcal{S}$ ▷ $B_c^{\mathcal{T}}$ and $B_c^{\mathcal{S}}$ are of the same class c.

      Compute $\mathcal{L}_c^{\mathcal{T}} = \frac{1}{|B_c^{\mathcal{T}}|} \sum_{(\boldsymbol{x},y)\in B_c^{\mathcal{T}}} \ell(\phi_{\boldsymbol{\theta}_t}(\boldsymbol{x}), y)$ and $\mathcal{L}_c^{\mathcal{S}} = \frac{1}{|B_c^{\mathcal{S}}|} \sum_{(\boldsymbol{s},y)\in B_c^{\mathcal{S}}} \ell(\phi_{\boldsymbol{\theta}_t}(\boldsymbol{s}), y)$

      Update $\mathcal{S}_c \leftarrow \texttt{opt-alg}_{\mathcal{S}}(D(\nabla_{\boldsymbol{\theta}}\mathcal{L}_c^{\mathcal{S}}(\boldsymbol{\theta}_t), \nabla_{\boldsymbol{\theta}}\mathcal{L}_c^{\mathcal{T}}(\boldsymbol{\theta}_t)), \varsigma_{\mathcal{S}}, \eta_{\mathcal{S}})$

    Update $\boldsymbol{\theta}_{t+1} \leftarrow \texttt{opt-alg}_{\boldsymbol{\theta}}(\mathcal{L}^{\mathcal{S}}(\boldsymbol{\theta}_t), \varsigma_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\theta}})$ ▷ Use the whole $\mathcal{S}$

Output: $\mathcal{S}$

Advantages of method:

- Does not require expensive unrolling of the recursive computation graph over the previous parameters $\{\theta_0, \ldots, \theta_{t-1}\}$
- Optimisation is significantly faster, memory efficient and scales up to the state-of-the-art deep neural networks
- Ablation study on hyper-parameters show method is not sensitive to varying hyper-parameters

Experiments:

- **Datasets**: MNIST, FashionMNIST, SVHN, CIFAR10
- **No. of synthetic images per class**: {1, 10, 50}
- **Neural networks:** MLP, ConvNet, LeNEt, AlexNet, VGG-11, ResNet-18
- **Activation functions:** sigmoid, ReLU, leaky ReLU
- **Pooling functions:** maxpool, average pool
- **Normalization functions:** batch, group, layer, instance norm
- **Applications:** continual learning, neural architecture search

Results of experiments:

- **Compared to coreset selection methods**
  - method outperforms all coreset selection methods (random, herding, K-Center, forgetting)
  - Results are better with MNIST and FashionMNIST than with CVHN and CIFAR10
- **Compared to Dataset Distillation**
  - Higher accuracy with fewer synthetic images per class
  - Smaller standard deviation over multiple runs
  - Faster training and requires less memory

Results of experiments:

- **Cross-architecture generalization**
  - Condensed images learned using one architecture performed well when used to train another architecture, particularly with convolutional networks
  - Best results obtained with ResNet generated images and ConvNet or ResNet as classifiers
- **Number of condensed images**
  - Increasing the number of condensed images improves the accuracies in all benchmarks and further closes gap with upper bound provided by training with original large dataset

Results of experiments:

- **Activation, normalization and pooling**
  - Leaky ReLU over ReLU and average pooling over max pooling for learning better condensed images as they allow for denser gradient flow
  - Instance normalization obtains better accuracies than its alternatives
- **Continual learning**
  - Condensed images are more data-efficient than images sampled by herding.
- **Neural Architecture Search**
  - Method gives highest testing performance and performance correlation compared to random sampling, herding and early-stopping
  - Method significantly decreases searching time and storage space compared to whole-dataset training and requires less training images than early stopping

Soft-Label Dataset Distillation and Text Dataset Distillation

Method: **Soft-Label Dataset Distillation (SLDD)**: Learn a synthetic set S - both the X and y, where y are “soft” class labels that achieves comparable generalization performance to training on the original set

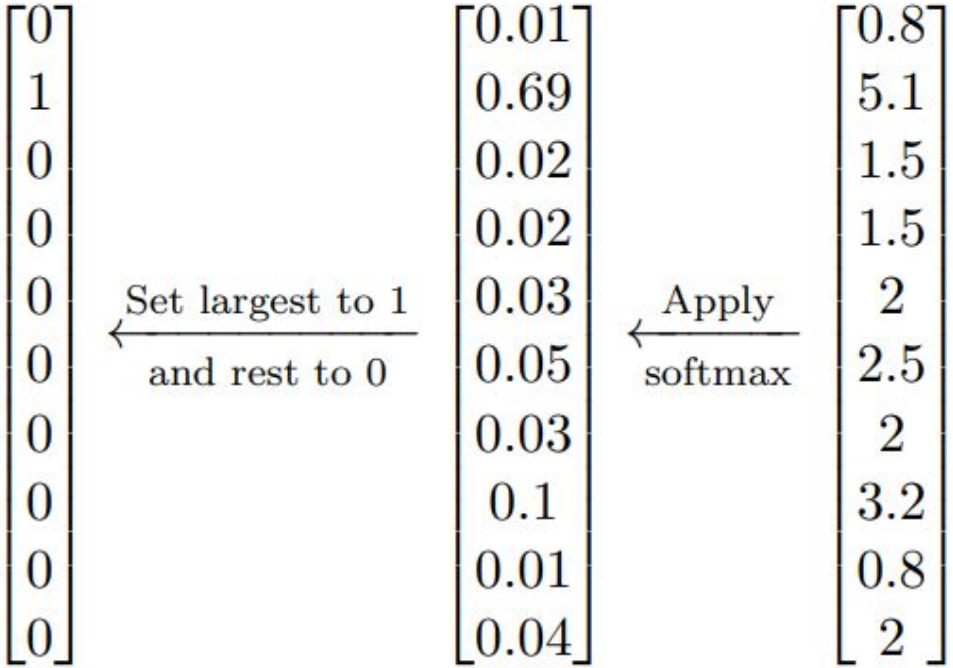
$$\tilde{\mathbf{x}}^*, \tilde{\mathbf{y}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\eta}} \mathcal{L}\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\eta}; \theta_0\right) = \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \ell\left(\mathbf{x}, \mathbf{y}, \theta_0 - \tilde{\eta} \nabla_{\theta_0} \ell\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \theta_0\right)\right)$$

x = predictor variable data
y = target variable soft labels
L = loss function

θ = parameters
η = learning rate of gradient descent
∇ = gradient

Key idea: any training image contains information about more than one class and using “soft” labels allows us to convey more information about the associated image

“Hard” labels vs “Soft” labels



“Hard” labels

“Soft” labels

Algorithm 1a Soft-Label Dataset Distillation (SLDD)

Input: $p(\theta_0)$: distribution of initial weights; M: the number of distilled data; α: step size; n: batch size; T: number of optimization iterations; $\tilde{y}_0$: initial value for $\tilde{y}$; $\tilde{\eta}_0$: initial value for $\tilde{\eta}$

Initialize distilled data
$\tilde{\mathbf{x}} = \{\tilde{x}_i\}_{i=1}^{M}$ randomly,
$\tilde{\mathbf{y}} = \{\tilde{y}_i\}_{i=1}^{M} \leftarrow \tilde{y}_0$,
$\tilde{\eta} \leftarrow \tilde{\eta}_0$
for each training step t = 1 to T **do**
  Get a mini-batch of real training data
  $(\mathbf{x}_t, \mathbf{y}_t) = \{x_{t,j}, y_{t,j}\}_{j=1}^{n}$
  One-hot encode the labels
  $(\mathbf{x}_t, \mathbf{y^*}_t) = \{x_{t,j}, \text{Encode}(y_{t,j})\}_{j=1}^{n}$
  Sample a batch of initial weights
  $\theta_0^{(j)} \sim p(\theta_0)$
  for each sampled $\theta_0^{(j)}$ **do**
    Compute updated model parameter with GD
    $\theta_1^{(j)} = \theta_0^{(j)} - \tilde{\eta} \nabla_{\theta_0^{(j)}} \ell\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \theta_0^{(j)}\right)$
    Evaluate the objective function on real training data: $\mathcal{L}^{(j)} = \ell\left(\mathbf{x}_t, \mathbf{y^*}_t, \theta_1^{(j)}\right)$
  end for
  Update distilled data
  $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} - \alpha \nabla_{\tilde{\mathbf{x}}} \sum_j \mathcal{L}^{(j)}$,
  $\tilde{\mathbf{y}} \leftarrow \tilde{\mathbf{y}} - \alpha \nabla_{\tilde{\mathbf{y}}} \sum_j \mathcal{L}^{(j)}$, and
  $\tilde{\eta} \leftarrow \tilde{\eta} - \alpha \nabla_{\tilde{\eta}} \sum_j \mathcal{L}^{(j)}$
end for

Output: distilled data $\tilde{\mathbf{x}}$; distilled labels $\tilde{\mathbf{y}}$; optimized learning rate $\tilde{\eta}$

Text Distillation - Modifying SLDD to apply to text sequences:

- It is difficult to use gradient methods directly on text data as it is discrete → **embed the text data into a continuous space by using pre-trained GloVe embeddings**
- If all sentences are padded/truncated to some pre-determined length, then each sentence is essentially just a one-channel image of size [length]∗ [embedding dimension]
- Since the distilled data produced by this algorithm would still be in the embedding space → **find the nearest sentences that correspond to the distilled embeddings**
    - for every column vector in the matrix, the nearest embedding vector from the original dictionary must be found. These embedding vectors must then be converted back into their corresponding words, and those words joined into a sentence.

Initialization of parameters:

- The samples created with fixed initialization θ_0 do not lead to high accuracies when the network is re-trained on them with a different initialization, as they contain information not only about the dataset but also about θ_0.
- Thus, the method is instead generalized to work with initializations randomly sampled from some restricted distribution → The resulting images appear to have much clearer patterns and much less random noise and the results show this method generalizes fairly well to other randomly sampled initializations from the same distribution.

$$\tilde{\mathbf{x}}^*, \tilde{\mathbf{y}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\eta}} \mathbb{E}_{\theta_0 \sim p(\theta_0)} \mathcal{L}\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\eta}; \theta_0\right)$$

Metrics used in evaluating model:

1. **Distillation ratio:** ratio of distillation accuracy to original accuracy. Since this is highly dependent on the number of distilled samples, M-sample distillation ratio notation used is

$$r_M = 100\% * \frac{[\text{distillation accuracy}]}{[\text{original accuracy}]}, M = [\text{number of distilled samples}]$$

2. **A% distillation size:** d_A = M where M is the minimum number of distilled samples required to achieve a distillation ratio of A%

Experiments:

<table>
<tr><th>SLDD (images)</th><th>TDD (text)</th></tr>
<tr><td colspan="2">Baselines for comparison: dataset distillation (DD), random real, optimised real, k-means, average real
Initialization of network: fixed initialization, random initialization (generated using the Xavier Initialization)</td></tr>
<tr><td>Datasets: MNIST, CIFAR10
Neural networks used: LeNet on MNIST, AlexCifarNet on CIFAR10, K-Nearest Neighbours on both</td><td>Datasets: IMDB sentiment analysis dataset, Stanford Sentiment Treebank 5-class task (SST5), Text Retrieval Conference question classification tasks with 6 (TREC6) and 50 (TREC50) classes
Neural networks used: TextConvNet, Bi-RNN, Bi-LSTM</td></tr>
</table>

Results of experiments:

SLDD (images)	TDD (text)
Fixed initialization: ● The SLDD algorithm produces images that result in equal or higher accuracies when compared to the original DD algorithm **Random initialization**: ● The distilled images produced in this way are more representative of the training data but generally result in lower accuracies when models are trained on them. ● Once again, images distilled using SLDD lead to higher distillation accuracies than DD when the number of distilled images is held constant.	**Fixed initialization**: ● The SLDD algorithm produces sentences that result in equal or higher accuracies when compared to the original DD algorithm. ● The TDD algorithm encourage the separation of classes when there are enough distilled samples to have one or more per class. **Random initialization**: ● Performance decreases as compared to fixed initialization. ● The difference in performance is larger for recurrent networks

Future work:

- Determine whether algorithm can be generalized to work with more variation in initializations (experiments used initialization from only one distribution)
- Determine whether single distilled dataset from algorithm can be used to train networks with different architectures (cross-architecture generalization)
- Explore label initialization methods: Whether it is better to separate similar classes (e.g. 3 and 8 in MNIST) to increase the network's ability to discern between them, or keep them together to allow soft-label information to be shared
- Use distilled datasets for speeding up Neural Architecture Search and other very compute-intensive meta-algorithms

Dataset Condensation with Differentiable Siamese Augmentation

Aim: Learn a synthetic set that can be effectively used with data augmentation to train deep neural networks, with randomly initialized parameters

Method: In addition to the gradient-matching method, apply the same randomly sampled data transformation to both sampled real and synthetic data at each training iteration

- allows for backpropagating the gradient of the loss function w.r.t. the synthetic data by differentiable data transformations.
- avoids averaging effect in a minibatch

$$\min_{\mathcal{S}} D(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{A}(\mathcal{S}, \omega^{\mathcal{S}}), \boldsymbol{\theta}_t), \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{A}(\mathcal{T}, \omega^{\mathcal{T}}), \boldsymbol{\theta}_t)),$$

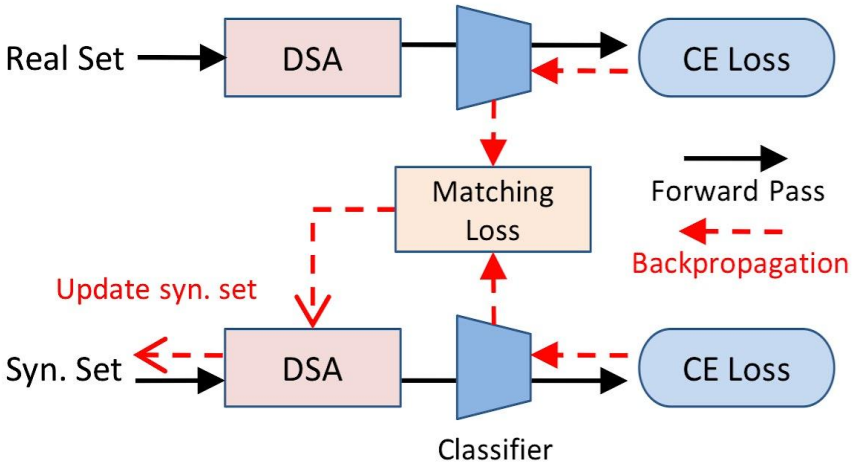
A = family of image transformations that preserve the semantics of the input, such as cropping, color jittering

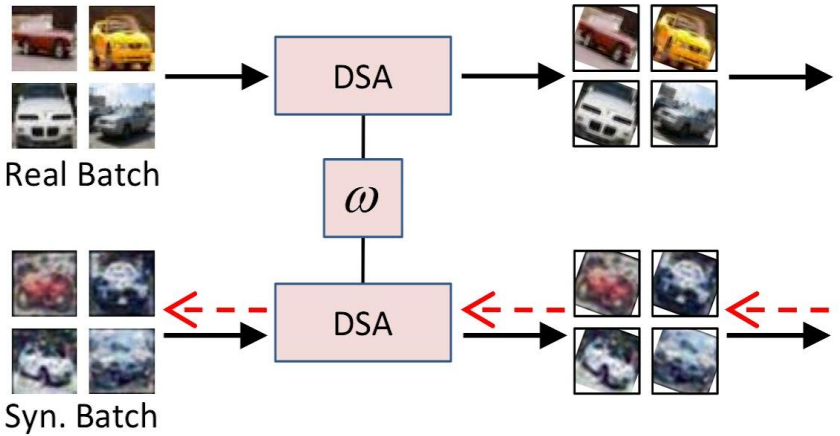
Theory behind why DSA works:

Hypothesis: Siamese augmentation acts as a strong regularizer on learned high-dimensional synthetic data S, alleviates its overfitting to the real training set

- synthetic set is forced to match the gradients from the real set under multiple transformations when sampled from the distribution
- renders the optimisation harder and less prone to overfitting

Flowchart of Method





Algorithm 1: Dataset condensation with differentiable Siamese augmentation.

Input: Training set $\mathcal{T}$

Required: Randomly initialized set of synthetic samples $\mathcal{S}$ for C classes, probability distribution over randomly initialized weights $P_{\boldsymbol{\theta}_0}$, deep neural network $\phi_{\boldsymbol{\theta}}$, number of training iterations K, number of inner-loop steps T, number of steps for updating weights $\varsigma_{\boldsymbol{\theta}}$ and synthetic samples $\varsigma_{\mathcal{S}}$ in each inner-loop step respectively, learning rates for updating weights $\eta_{\boldsymbol{\theta}}$ and synthetic samples $\eta_{\mathcal{S}}$, differentiable augmentation $\mathcal{A}_\omega$ parameterized with ω, augmentation parameter distribution Ω, random augmentation $\mathcal{A}$.

for $k = 0, \cdots, K-1$ **do**
  Initialize $\boldsymbol{\theta}_0 \sim P_{\boldsymbol{\theta}_0}$
  for $t = 0, \cdots, T-1$ **do**
    for $c = 0, \cdots, C-1$ **do**
      Sample $\omega \sim \Omega$ and a minibatch pair $B_c^{\mathcal{T}} \sim \mathcal{T}$ and $B_c^{\mathcal{S}} \sim \mathcal{S}$ ▷ $B_c^{\mathcal{T}}$, $B_c^{\mathcal{S}}$ are of class c.
      Compute $\mathcal{L}_c^{\mathcal{T}} = \frac{1}{|B_c^{\mathcal{T}}|} \sum_{(\boldsymbol{x},y)\in B_c^{\mathcal{T}}} \ell(\phi_{\boldsymbol{\theta}_t}(\mathcal{A}_\omega(\boldsymbol{x})), y)$ and $\mathcal{L}_c^{\mathcal{S}} = \frac{1}{|B_c^{\mathcal{S}}|} \sum_{(\boldsymbol{s},y)\in B_c^{\mathcal{S}}} \ell(\phi_{\boldsymbol{\theta}_t}(\mathcal{A}_\omega(\boldsymbol{s}), y)$
      Update $\mathcal{S}_c \leftarrow \mathtt{sgd}_{\mathcal{S}}(D(\nabla_{\boldsymbol{\theta}}\mathcal{L}_c^{\mathcal{S}}(\boldsymbol{\theta}_t), \nabla_{\boldsymbol{\theta}}\mathcal{L}_c^{\mathcal{T}}(\boldsymbol{\theta}_t)), \varsigma_{\mathcal{S}}, \eta_{\mathcal{S}})$
    Update $\boldsymbol{\theta}_{t+1} \leftarrow \mathtt{sgd}_{\boldsymbol{\theta}}(\mathcal{L}(\boldsymbol{\theta}_t, \mathcal{A}(\mathcal{S})), \varsigma_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\theta}})$ ▷ Use $\mathcal{A}$ for the whole $\mathcal{S}$

Output: $\mathcal{S}$

Advantages of method:

- ability to exploit the information in real training images more effectively by augmenting them in several ways, transfer this augmented knowledge to synthetic images
- sharing the same transformation across real and synthetic images allows synthetic images to learn certain prior knowledge in real images
- once synthetic images are learned, they can be used with various data augmentation strategies to train different neural network architectures

Experiments:

- **Datasets**: MNIST, FashionMNIST, SVHN, CIFAR10, CIFAR100
- **Network architectures:** MLP, ConvNet, LeNet, AlexNet, VGG-11, ResNet-18
- **Initialization:**
  - Network parameters: Kaiming initialization
  - labels of synthetic data: pre-defined evenly for all classes
  - Synthetic images: randomly sampled real images of corresponding class, random noise
- **No. of synthetic images per class**: {1, 10, 50}
- **Baselines:**
  - **Coreset selection:** random, herding, forgetting
  - **Training set synthesis:** dataset distillation, label distillation, dataset condensation

Results of Experiments:

10 Category datasets:

- Method achieves the best performance in most settings and in particular obtains significant gains when learning 10 and 50 images/class
- Method obtains comparable or worse performance than **dataset condensation** in case of 1 image/class → method acts as a regularizer on DC, as the synthetic images are forced to match the gradients from real training images under different transformations. Thus it is expected that the method works better when the solution space (synthetic set) is larger.
- Performance gap between upperbound accuracy (trained with whole set) and accuracy obtained by synthetic images increases when task is more challenging (CIFAR10 > MNIST)
- Synthetic images are easily recognizable and more similar to real ones than ones generated by previous methods

Results of Experiments:

Cross-Architecture Generalization:

- Synthetic images learned by the convolutional architectures (ConvNet, LeNet, AlexNet, VGG and ResNet) perform best and generalizes to the other convolutional ones, while the MLP network produces less informative synthetic images overall
- The most competitive architecture, ResNet provides the best results when trained as a classifier on the synthetic images.

Effectiveness of augmentation:

- proposed Siamese augmentation always achieves the best performance when used with individual augmentation. The largest improvement is obtained by applying Siamese augmentation with cropping.
- Applying all the augmentations (color jittering, cropping, cutout, flipping, scaling, rotation) improve the performance on CIFAR10 compared to using None.

Results of Experiments:

Continual Learning:

- Tasks are incrementally learned on three digit recognition datasets, goal is to preserve the performance in the seen tasks while learning new ones
- DSA always outperforms the other two memory construction methods (herding, and DC)

Neural Architecture Search:

- Proxy set of synthetic images achieves strongest correlation
- Time cost of implementing NAS on proxy set is only 1.2% of implementing NAS on the whole dataset
- INdicates DSA can speed up NAS

Generative Teaching Networks: Accelerating Neural Architecture Search by Learning to Generate Synthetic Training Data

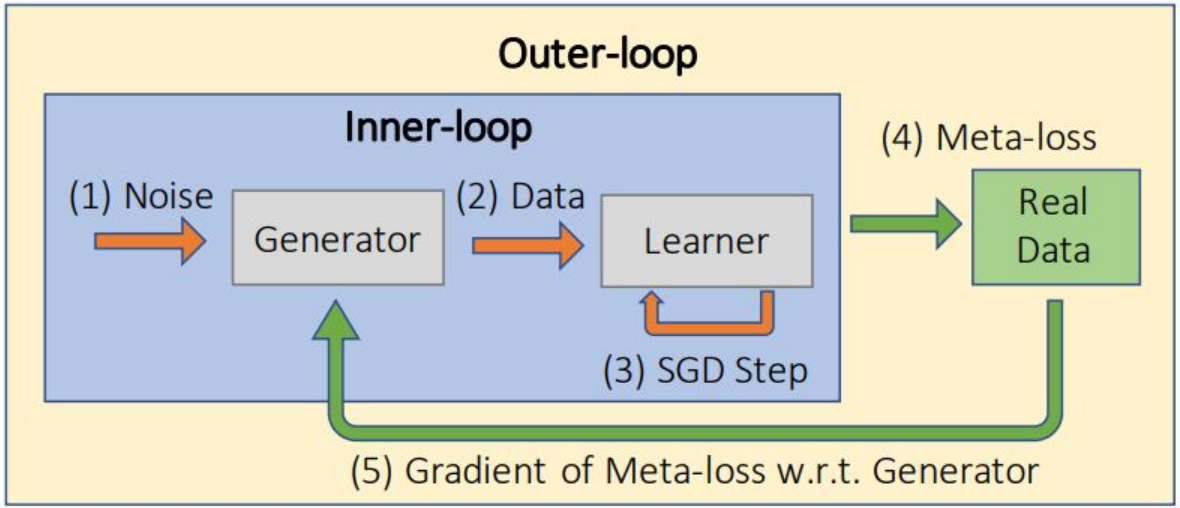
Aim:

- accelerate neural architecture search - what matters is how well and inexpensively we can identify architectures that achieve high asymptotic accuracy when later trained on the full (real) training set –> being able to train architectures rapidly - with very few SGD steps (As opposed to other techniques which aim to produce high test accuracy with synthetic images)
- Learner agnostic - Synthetic data generated is not only agnostic to the weight initialization of the learner network, but is also agnostic to the learner's architecture

Method:

Train a data-generating network such that a learner network trained on its data produces high accuracy in a target task.

- Two nested training loops: an inner loop to train a learner network, and an outer-loop to train a generator network that produces synthetic training data for the learner network
- While computing the gradients for the generator, compute the gradients of hyperparameters of the inner-loop SGD update rule (its learning rate and momentum), which are updated after each outer-loop at no additional cost.
- To reduce memory requirements, gradient-checkpointing is leveraged when computing meta-gradients.



1. Generator takes noise as input and generates synthetic data
2. Weights of learner are updated using the synthetic data (using SGD or Adam etc)
3. Updated weights from learner are used on real data and loss is calculated
4. Gradient of loss wrt generator data is calculated (backpropagation) and used to update generator

Weight normalization:

- stabilize meta-gradient training by normalizing the generator and learner weights
- Instead of updating the weights (W) directly, parameterize them as

$$W = g \cdot V / || V ||$$

  and instead update the scalar g and vector V .

- **Hypothesize that weight normalization will also stabilize and improve other meta-gradient techniques, although future work is required to test this hypothesis in meta-learning contexts besides GTNs**

Learning a Curriculum:

- A curriculum can be encoded as a series of input vectors to the generator (i.e. instead of sampling the z_t inputs to the generator from a Gaussian distribution, a sequence of z_t inputs can be learned).
- A curriculum can thus be learned by differentiating through the generator to optimize this sequence (in addition to the generator's parameters).

Experiments:

- **Weight normalization**: with and without
- **Learning a Curriculum:** noise, all shuffled, shuffled batch, full curriculum
- **Neural Architecture Search**
- **Datasets:** MNIST, CIFAR10

Results of Experiments:

- weight normalization significantly improves the stability of meta-learning —> future work
- training with synthetic data is more effective when learning such data jointly with a curriculum that orders its presentation to the learner
- GTNs can generate a synthetic training set that enables more rapid learning in a few SGD steps than real training data in two supervised learning domains (MNIST and CIFAR10) and in a reinforcement learning domain

Flexible Dataset Distillation: Learn Labels Instead of Images

Method: Craft synthetic labels for arbitrarily chosen standard images

$$\tilde{\boldsymbol{Y}}_{\mathcal{S}}^{*} = \arg\min_{\tilde{\boldsymbol{Y}}_{\mathcal{S}}} \sum_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{T}} L\left(f_{\boldsymbol{\Theta}'}\left(\boldsymbol{x}\right),\boldsymbol{y}\right), \quad \text{with} \quad \boldsymbol{\Theta}' = \boldsymbol{\Theta} - \alpha\nabla_{\boldsymbol{\Theta}} \sum_{\tilde{\boldsymbol{x}},\tilde{\boldsymbol{y}}\sim\mathcal{S}} L\left(f_{\boldsymbol{\Theta}}\left(\tilde{\boldsymbol{x}}\right),\tilde{\boldsymbol{y}}\right),$$

Advantages of method:

- Exploits the data statistics of natural images and the lower-dimensionality of labels compared to images as parameters for meta-learning
- Enables cross-dataset knowledge distillation
- More flexible distilled dataset that is better transferable across optimizers, architectures, learning iterations
- Does not rely on second-order gradients. Instead, method is based on convex optimization layers that avoids high-order gradients and provides better optimization

Theory:

Algorithm:

- One option would be to simulate the whole training procedure for multiple gradient steps within the inner loop. However, this requires back-propagating through a long inner loop, and ultimately requires a fixed training schedule with optimized learning rates for strong performance.
- In order to produce a dataset that can be used in a standard training pipeline downstream, modify this to **perform gradient descent iteratively on the model and the distilled labels, rather than performing many inner (model) steps for each outer (dataset) step**
- This increases efficiency significantly due to a shorter compute graph for backpropagation.

Theory:

Overfitting:
When there are very few training examples, the model converges quickly to an overfitted local minimum

To overcome this problem:

- Detect overfitting when it occurs, reset the model to a new random initialization and keep training.
- Measure the moving average of target problem accuracy, and when it has not improved for set number of iterations, reset the model.

This periodic reset of the model after varying number of iterations is helpful for learning labels that are useful for all stages of training and less sensitive to the number of iterations used.

Algorithm 1 Label distillation with ridge regression (RR)

Input: $\mathcal{S}$: synthetic set with N initially unlabelled base examples $\tilde{\boldsymbol{X}}_{\mathcal{S}}$; $\mathcal{T}$: labelled target set with real training examples; β: step size; α: pseudo-gradient step size; N_o, N_i: outer (resp. inner) loop batch size; C: number of classes in the target set; λ: RR regularization parameter
Output: distilled labels $\tilde{\boldsymbol{Y}}_{\mathcal{S}}$ and a reasonable number of training steps T_i
$\tilde{\boldsymbol{Y}}_{\mathcal{S}} \leftarrow \mathbf{1}/C$ //Uniformly initialize synthetic labels
$\boldsymbol{\Theta} \sim p(\boldsymbol{\Theta})$ //Randomly initialize feature extractor parameters
$\boldsymbol{W} \leftarrow \mathbf{0}$ //Initialize global RR classifier weights
while $\tilde{\boldsymbol{Y}}_{\mathcal{S}}$ not converged **do**
  $\left(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}}\right) \sim \mathcal{S} = \left(\tilde{\boldsymbol{X}}_{\mathcal{S}}, \tilde{\boldsymbol{Y}}_{\mathcal{S}}\right)$ //Sample a minibatch of N_i synthetic examples
  $(\boldsymbol{X}, \boldsymbol{Y}) \sim \mathcal{T}$ //Sample a minibatch of N_o real training examples
  Calculate $\boldsymbol{W}_l$ using Eq. 3 //Calculate current minibatch RR classifier weights
  $\boldsymbol{W} \leftarrow (1-\alpha)\boldsymbol{W} + \alpha \boldsymbol{W}_l$ //Update global RR classifier weights
  $\tilde{\boldsymbol{Y}}_{\mathcal{S}} \leftarrow \tilde{\boldsymbol{Y}}_{\mathcal{S}} - \beta \nabla_{\tilde{\boldsymbol{Y}}_{\mathcal{S}}} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in (\boldsymbol{X},\boldsymbol{Y})} L\left(f_{\boldsymbol{W} \circ \boldsymbol{\Theta}}(\boldsymbol{x}), \boldsymbol{y}\right)$ //Update synthetic labels
  $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \beta \nabla_{\boldsymbol{\Theta}} \sum_{(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{y}}) \in (\tilde{\boldsymbol{X}},\tilde{\boldsymbol{Y}})} L\left(f_{\boldsymbol{W} \circ \boldsymbol{\Theta}}(\tilde{\boldsymbol{x}}), \tilde{\boldsymbol{y}}\right)$ //Update feature extractor
  if $\mathbb{E}_{\boldsymbol{x},\boldsymbol{y} \sim \mathcal{T}}\left[L\left(f_{\boldsymbol{W} \circ \boldsymbol{\Theta}}(\boldsymbol{x}), \boldsymbol{y}\right)\right]$ did not improve **then**
    $\boldsymbol{\Theta} \sim p(\boldsymbol{\Theta})$ //Reset feature extractor
    $\boldsymbol{W} \leftarrow \mathbf{0}$ //Reset global RR classifier weights
    $T_i \leftarrow$ iterations since previous reset //Record time to overfit
  end if
end while

Second-order version:

- Consists both inner and outer loop
- Inner loop: one update of the model parameters. Then, backpropagate to update the synthetic labels (using real training set examples). Normalize labels to represent valid probability distribution.
- Outer loop: update model parameters after updating synthetic labels (using synthetic labels and base examples)

First-order version with ridge regression:

- Use pseudo-gradient generated via closed form solution to ridge-regression
- Use a RR layer as the final output layer of base network - original model is decomposed into feature extractor θ and RR classifier W.
- RR solves for optimal local weights W_l that classify the features of current minibatch examples. Exploit this local minibatch solution by taking a pseudo-gradient step that updates global weights. Then, update the synthetic labels by back-propagating through local weights W_l.

Experiments:

- **Datasets:**
  - **Within-dataset distillation:** MNIST, CIFAR-10, CIFAR-100
  - **Cross-dataset distillation:** EMNIST, KMNIST< Kuzushiji-49, MNIST, CUB, CIFAR-10
- **Network Architectures**
  - LeNet for MNIST and similar
  - AlexNet for CIFAR-10, CIFAR-100, CUB

Results of Experiments:

- **Within-Dataset Distilation**
  - Outperforms DD and SLDD on MNIST, in part due to LD enabling the use of more steps. Outperforms DD on CIFAR-10.
  - LD scales to a significantly larger number of classes than 10 by application to CIFAR-100 - method leads to clear improvements over baseline
- **Cross-Dataset Distillation:**
  - EMNIST → MNIST
  - EMNIST → Kuzushiji-MNIST
  - EMNIST → Kuzushiji-49
  - CUB → CIFAR-10
  - Able to distil labels on examples of a different source dataset and achieve surprisingly good performance on the target problem, given no target data is used when training these models.
  - It is indeed possible to distill the knowledge of one dataset into base examples from a different but related dataset through crafting synthetic labels
  - RR approach surpasses second-order method

Results of Experiments:

- **Flexibility of distilled datasets**
  1) **Number of training steps:** LD is relatively insensitive to number of steps used to train a model on distilled data, as compared to DD and SLDD
  2) **Optimisation parameters:** original DD relies on training distilled data in fixed sequence and changing the order of examples leads to large decrease in accuracy. LD does not depend on specific order of examples and can be used with off-the-shelf optimisers such as Adam
- **Cross-Architecture Generalization**
  - Using AlexNet to distil labels and training AlexNet, LeNet and ResNet-18 using distilled labels, results show labels are transferable both within and across dataset distillation scenarios. LD has greater transferability than DD.

Possible Future work:

Understanding the impact of dataset distillation on any underlying biases in the data

Mnemonics Training Multi-Class Incremental Learning without Forgetting

Aim: Multi-Class Incremental Learning - extract exemplars for each class to be kept around when the model advances to the next training phase, so as not to retain all data from previous training phases

Method:

- parametrize the exemplars using image-size parameters, optimise them in an end-to-end scheme.
- The model in each phase can not only learn the optimal exemplars from new class data, but also adjust the exemplars of previous phases to fit the current data distribution.
- Use **bilevel optimisation program (BOP)** that alternates the learning of two levels of models

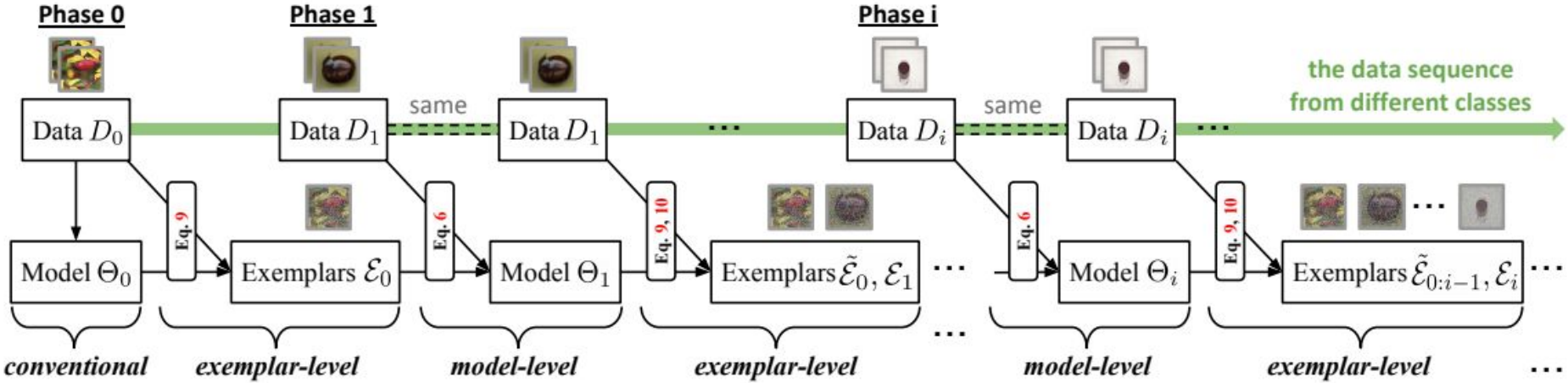


Figure 2. The computing flow of the proposed *mnemonics* training. It is a global BOP that alternates the learning of *mnemonics* exemplars (we call *exemplar-level* optimization) and MCIL models (*model-level* optimization). The *exemplar-level* optimization within each phase is detailed in Figure 3. $\tilde{\mathcal{E}}$ denotes the old exemplars adjusted to the current phase.

1. Temporary model is trained with exemplars as input
2. Validation loss on new class data is computed and the gradients are back-propagated to optimise the input layer
3. Iterate steps 1-2 to derive representative exemplars for later training phases

Global Level BOP

The classification model is incrementally trained in each phase on the union of new class data and old class mnemonics exemplars. In turn, based on this model, the new class mnemonics exemplars (i.e., the parameters of the exemplars) are trained before omitting new class data.

$$\min_{\Theta_i} \mathcal{L}_c(\Theta_i; \mathcal{E}^*_{0:i-1} \cup D_i) \tag{4a}$$

$$\text{s.t. } \mathcal{E}^*_{0:i-1} = \arg\min_{\mathcal{E}_{0:i-1}} \mathcal{L}_c\big(\Theta_{i-1}(\mathcal{E}_{0:i-1}); \mathcal{E}_{0:i-2} \cup D_{i-1}\big),$$

Model Level

Mnemonics exemplars $\varepsilon_{0:1\text{-}1}$ as part of the input and previous $\Theta_{i\text{-}1}$ as the model initialization.

$$\mathcal{L}_{\text{all}} = \lambda \mathcal{L}_c(\Theta_i; \mathcal{E}_{0:i-1} \cup D_i) + (1-\lambda)\mathcal{L}_d(\Theta_i; \Theta_{i-1}; \mathcal{E}_{0:i-1} \cup D_i),$$

$$\Theta_i \leftarrow \Theta_i - \alpha_1 \nabla_\Theta \mathcal{L}_{\text{all}}.$$

Then, Θi will be used to train the parameters of the mnemonics exemplars, i.e., to solve the exemplar-level problem

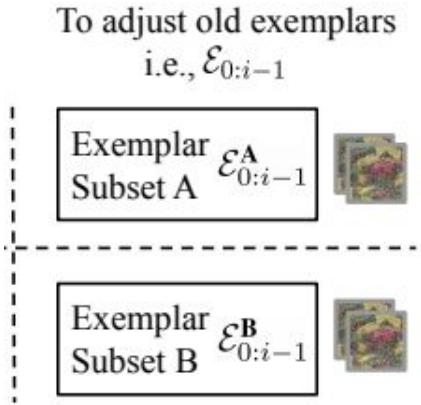
Exemplar Level

1. The image-size parameters of ε_i are initialized by a random sample subset S of D_i.
2. Initialize a temporary model Θ_i' using Θ_i and train Θ_i' on ε_i for a few iterations by gradient descent
3. As the Θ# i and Ei are both differentiable, we are able to compute the loss of Θ_i' on D_i, and back-propagate this validation loss to optimize ε_i.

$$\min_{\mathcal{E}_i} \mathcal{L}_c\big(\Theta_i'(\mathcal{E}_i); D_i\big)$$

$$\text{s.t. } \Theta_i'(\mathcal{E}_i) = \arg\min_{\Theta_i} \mathcal{L}_c(\Theta_i; \mathcal{E}_i).$$

Adjusting $\varepsilon_{0:1\text{-}1}$



- Split $\varepsilon_{0:1\text{-}1}$ into two subsets and subject to $\varepsilon_{0:1\text{-}1} = \varepsilon^{A}_{0:1\text{-}1} \cup \varepsilon^{B}_{0:1\text{-}1}$.
- Use one of them, e.g. $\varepsilon^{B}_{0:1\text{-}1}$, as the validation set (i.e., a replacement of $D_{0:i-1}$) to optimize the other one, e.g., $\varepsilon^{A}_{0:1\text{-}1}$

$$\mathcal{E}^{\mathbf{A}}_{0:i-1} \leftarrow \mathcal{E}^{\mathbf{A}}_{0:i-1} - \beta_2 \nabla_{\mathcal{E}^{\mathbf{A}}} \mathcal{L}_c\left(\Theta'_i(\mathcal{E}^{\mathbf{A}}_{0:i-1}); \mathcal{E}^{\mathbf{B}}_{0:i-1}\right),$$

$$\mathcal{E}^{\mathbf{B}}_{0:i-1} \leftarrow \mathcal{E}^{\mathbf{B}}_{0:i-1} - \beta_2 \nabla_{\mathcal{E}^{\mathbf{B}}} \mathcal{L}_c\left(\Theta'_i(\mathcal{E}^{\mathbf{B}}_{0:i-1}); \mathcal{E}^{\mathbf{A}}_{0:i-1}\right),$$

Experiments:

- **Datasets:** CIFAR-100, ImageNet
- **Network Architectures:** 32-Layer ResNet for CIFAR-100, 18-Layer ResNet for IMageNet, weight transfer operations to reduce forgetting between models in adjacent phases
- **Number of incremental phases:** {5, 10, 25}
- **Benchmark protocol:** LUCIR

Results of Experiments:

- taking our mnemonics training as a plug-in module on the state of-the-art and other baseline architectures consistently improves their performance. Mnemonics is greatly helpful in reducing forgetting problems for every method.
- the boost by mnemonics training becomes larger in more-phase settings, as model is not overfitted to herding exemplars mnemonics- exemplars are given both strong optimizability and flexible adaptation ability through BOP
- Ablation study: approach achieves the highest average accuracies and the lowest forgetting rates in all settings.

DATASET META-LEARNING FROM KERNEL RIDGE-REGRESSION

Method:

1. **ε-approximation**: theoretical framework for understanding dataset distillation and compression
2. **Kernel Inducing Points (KIP)**: meta-learning algorithm for obtaining ε-approximation of datasets
3. **Label Solve (LS)**: variant of KIP that gives closed-form solution for obtaining distilled datasets differing only via labels

Motivation for using kernel ridge-regression:

1. KRR performs convex-optimisation resulting in a closed-form solution → when optimising for the training parameters of KRR, we only have to consider first-order optimisation
2. Since KRR for a neural tangent kernel (NTK) approximates the training of the corresponding wide neural network, we expect the use of neural kernels to yield ε-approximations of D for learning algorithms given by a broad class of neural networks trainings as well

ε-approximation:

Definition 2. Fix a loss function ℓ and let $f, \tilde{f} : \mathbb{R}^d \to \mathbb{R}^C$ be two functions. Let $\epsilon \geq 0$.

1. Given a distribution $\mathcal{P}$ on $\mathbb{R}^d \times \mathbb{R}^C$, we say f and $\tilde{f}$ are *weakly ϵ-close* with respect to $(\ell, \mathcal{P})$ if
$$\left|\mathbb{E}_{(x,y)\sim\mathcal{P}}\Big(\ell(f(x), y)\Big) - \mathbb{E}_{(x,y)\sim\mathcal{P}}\Big(\ell(\tilde{f}(x), y)\Big)\right| \leq \epsilon. \tag{1}$$
2. Given a distribution $\mathcal{P}$ on $\mathbb{R}^d$ we say f and $\tilde{f}$ are *strongly ϵ-close* with respect to $(\ell, \mathcal{P})$ if
$$\mathbb{E}_{x\sim\mathcal{P}}\Big(\ell(f(x), \tilde{f}(x))\Big) \leq \epsilon. \tag{2}$$

Definition 3. Fix learning algorithms A and $\tilde{A}$. Let D and $\tilde{D}$ be two labeled datasets in $\mathbb{R}^d$ with label space $\mathbb{R}^C$. Let $\epsilon \geq 0$. We say $\tilde{D}$ is a *weak ϵ-approximation* of D with respect to $(\tilde{A}, A, \ell, \mathcal{P})$ if $\tilde{A}_{\tilde{D}}$ and A_D are weakly ϵ-close with respect to $(\ell, \mathcal{P})$, where ℓ is a loss function and $\mathcal{P}$ is a distribution on $\mathbb{R}^d \times \mathbb{R}^C$. We define *strong ϵ-approximation* similarly. We drop explicit reference to (some of) the $\tilde{A}, A, \ell, \mathcal{P}$ if their values are understood or immaterial.

Algorithm 1: Kernel Inducing Point (KIP)

Require: A target labeled dataset (X_t, y_t) along with a kernel or family of kernels.

1: Initialize a labeled support set (X_s, y_s).
2: **while** not converged **do**
3: Sample a random kernel. Sample a random batch $(\bar{X}_s, \bar{y}_s)$ from the support set. Sample a random batch $(\bar{X}_t, \bar{y}_t)$ from the target dataset.
4: Compute the kernel ridge-regression loss given by (7) using the sampled kernel and the sampled support and target data.
5: Backpropagate through $\bar{X}_s$ (and optionally $\bar{y}_s$ and any hyper-parameters of the kernel) and update the support set (X_s, y_s) by updating the subset $(\bar{X}_s, \bar{y}_s)$.
6: **end while**
7: **return** Learned support set (X_s, y_s)

KIP: optimise with respect to X_s

LS: optimise with respect to y_s

$$L(X_s, y_s) = \frac{1}{2} \| y_t - K_{X_t X_s} (K_{X_s X_s} + \lambda I)^{-1} y_s \|^2,$$

KIP variations:

1. Randomly augment the sampled target batches in KIP. This effectively enhances the target dataset (X_t, y_t), and we obtain improved results in this way, with no extra computational cost with respect to the support size.
2. Choose a corruption fraction $0 \leq \rho < 1$ and do the following:
    a. Initialize a random ρ-percent of the coordinates of each support datapoint via some corruption scheme (zero out all such pixels or initialize with noise).
    b. Do not update such corrupted coordinates during the KIP training algorithm (i.e. we only perform gradient updates on the complementary set of coordinates)
    c. Call this resulting algorithm KIPρ. In this way, KIPρ is ρ-corrupted → used to obtain highly corrupted datasets

- **Datasets**: MNIST, CIFAR-10, additionally FashionMNIST for LS

Experiments	**Results**
Single Kernel: Investigates optimising KIP and LS for compressing datasets and achieving state of the art performance for individual kernels	Using RBF and FC1 kernels and with KRR training, ● State of the art performance in terms of accuracy and number of images required, rival performance of deep CNNs ● Fewer KIP images perform on par with tens or hundreds times more natural images
	Using RBF and FC1 kernels and with neural network trainings: ● FC1 trained on KIP images outperform prior methods ● KIP, when scaled up to deeper architectures, should continue to yield strong neural network performance
	Using NGNP and NTK kernels associated to FC1, Myrtle-5, Myrtle-10 and with KRR training: ● The more targets are used, the better the performance ● When all possible targets are used, obtain optimal compression ratio of roughly one order of magnitude at intermediate support sizes

Experiments	Results
Kernel to Kernel: Investigates transferability of learned datasets across different kernels	Draw Kernels coming from fully connected and CNN layers of depths 1-3. ● KIP-datasets trained with random sampling of all six kernels do better on average than KIP-datasets trained using individual kernels ● For LS, transferability between FC1 and MyrtFor LS, transferability between FC1 and Myrtle-10 kernels on CIFAR-10 is highly robust. There is only a negligible difference in performance in nearly all instances between data with transferred learned labels and with natural labels.
Kernel to Neural Networks: Investigates transferability of KIP-learned datasets to training neural networks	KIP-learned datasets, even with heavy corruption, transfer remarkably well to the training of neural networks. ● The deterioration in test accuracy for KIP-images is limited as a function of the corruption fraction ● corrupted KIP-images typically outperform uncorrupted natural images ● sometimes higher corruption leads to better test performance ● robust to both dataset size and hyperparameters ● KIP-learned images can be useful in accelerating hyperparameter search