Refurbish Your Training Data : Reusing Partially Augmented Samples for Faster Deep Neural Network Training

- Submitted conference/journel name/years: Published as a conference paper at USENIX, 2021.
- Author: Gyewon Lee, Irene Lee, Hyeonmin Ha et al.
- Presented date: 2022.05.09.

Presented by: JunYeong Park

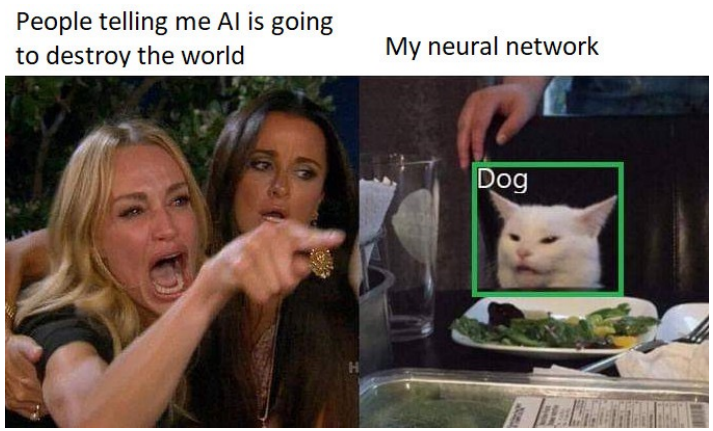
Student ID: 72220216

Content

Introduction

1. Generalization and Augmentation

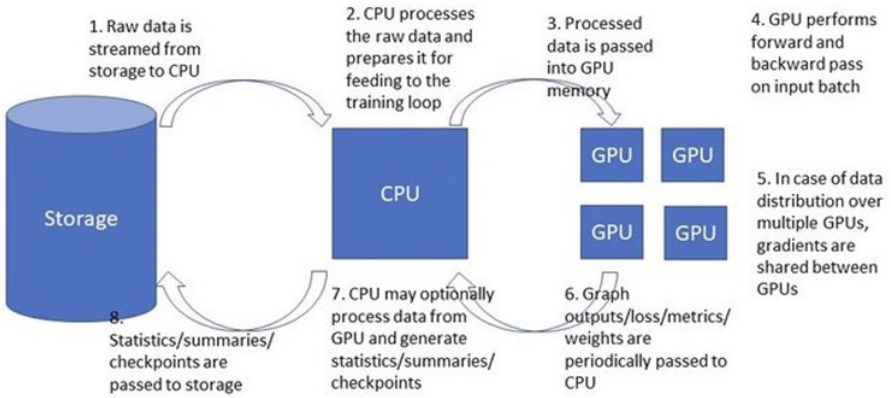
- Generalization: model's ability to adapt properly to new, previously unseen data, drawn from the same distribution as the one used to create the model.
- Data augmentation has been used in DNN training to improve generalization of DL models in many domains. It provides additional samples to model training → improve model generalization

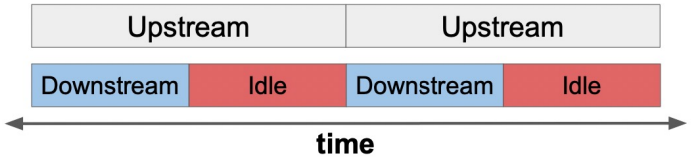


Introduction

2. Training Pipeline and CPU Overhead

- Storage (Data store) → CPU (Data Preprocessing) → GPU (Gradient Computation)





- Data augmentation consists of a sequence of transformation layers → Computational expensive

- The GPU/TPU is getting faster, but the CPU's data processing speed cannot keep up with the gradient calculation speed of the accelerators.

  ➔ GPU idle time = CPU bottleneck(overhead)

Introduction

3. Limitations of Existing Approaches

1) HW Accelerators

- HW Accelerators are optimized for massive parallel execution of homogenous operations.

  (like gradient computation, identical and deterministic computations to each training samples)

- However, Data augmentation is stochastic operations to each sample in a random fashion.

- Data augmentation and gradient computation may not be parallelized.

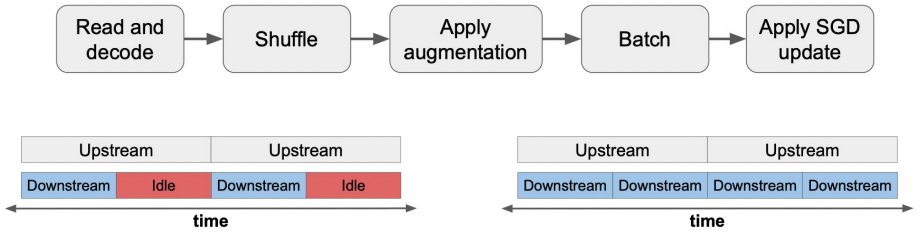
Introduction

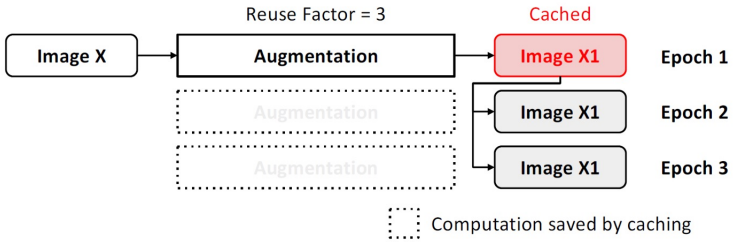
3. Limitations of Existing Approaches

2) Data Echoing

- Split the pipeline into upstream and downstream based on the expensive part of the calculation and reusing the data generated by the upstream a certain number of times in order to reclaim idle capacity.

  ➔ Contribution: Reduces model training time by converging the same model performance with less fresh data.





- (Re-augmentation overhead) If a sample is reused before augmentation, the reused sample needs to be re-augmented, and thus the overhead from data augmentation remains the same.

- (Diversity problem, lack of unique sample) When fully augmented samples are simply reused for gradient computation, the # of unique augmented samples significantly decreases and it degrades the model performance.

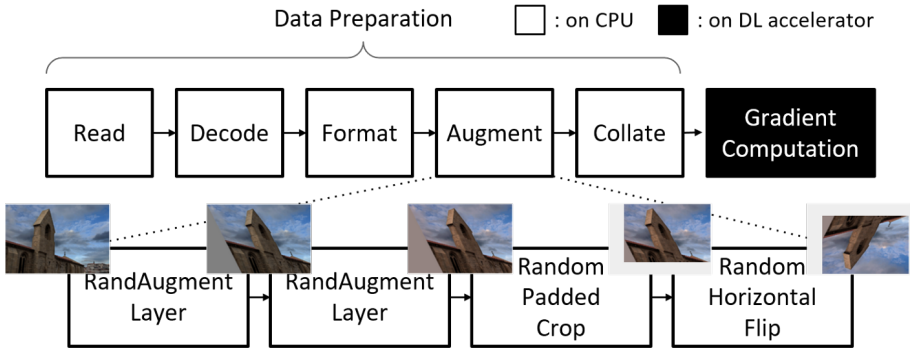
Data Refurbishing

1. **Partial Augmentation + Final Augmentation = Full Augmentation**

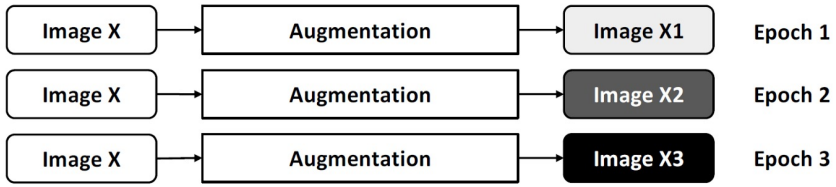
- Partial Augmentation: the first few transformations in the full augmentation pipeline
- Final Augmentation: the rest of the augmentation pipeline

2. Two Configuration

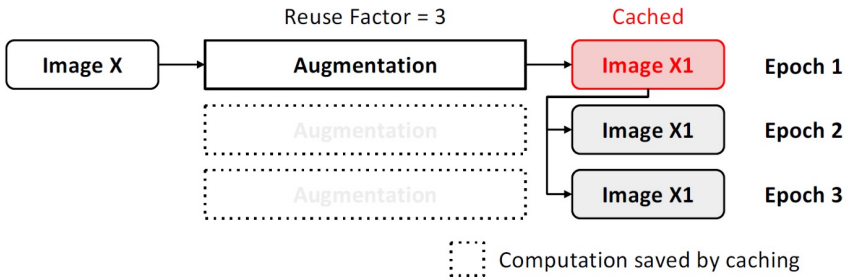
- **Reused factor 'r'**: how many times to reuse each cached sample (typically smaller than 5)
- **Split Policy**: how to split the full augmentation pipeline into the partial and final augmentations (# of strategies = # of augmentation layers, not exceed 20 even in extreme cases)



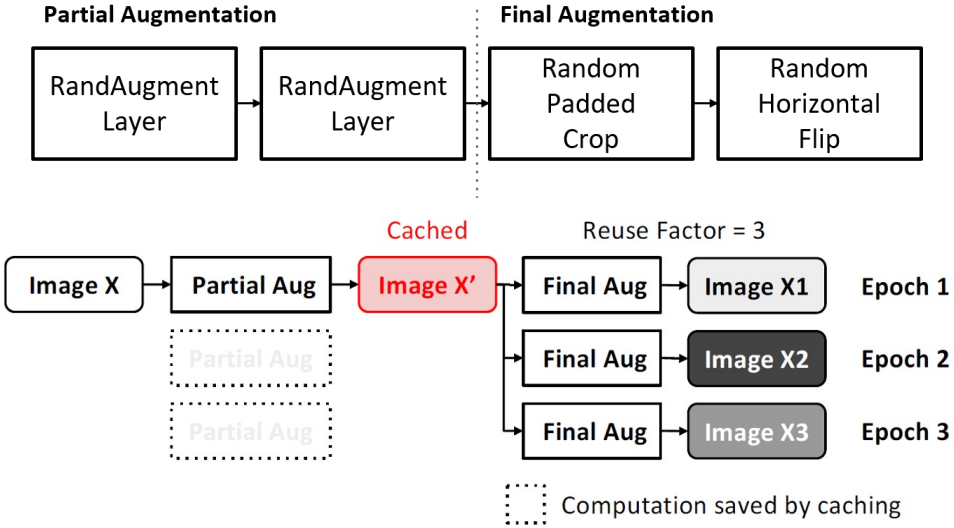
Data Refurbishing



Standard data augmentation



Data echoing



Data Refurbishing

In data refurbishing,

partial augmented samples are cached, reused r-times and renewed to preserve sample diversity.

Data Refurbishing

3. Problem Formulation (Mathematically explanation of preserving sample diversity)

- $\mathcal{X}$ and $\mathcal{X}'$: the sample space before and after augmentation, respectively
- Augmentation A: a finite set of functions $A := \{f_1, f_2, \ldots, f_{|A|}\}$ for $\forall_i f_i : \mathcal{X} \to \mathcal{X}'$.
- Partial Aug. A_P, Final Aug. A_F of A satisfy $\{f_F \circ f_P | f_P \in A_P, f_F \in A_F\} = A$.

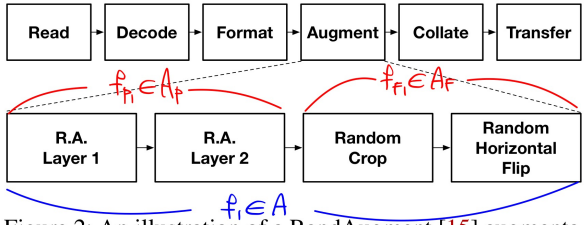


Figure 2: An illustration of a RandAugment [15] augmentation pipline in a typical data preparation pipeline. R.A. Layer stands for a RandAugment layer.

4. Assumption

- Discrete Uniform Distribution: For all augmentation A, the probability of choosing $f \in A$ is uniform $\forall_{f \in A} P(f) = \frac{1}{|A|}$
- **Balanced Eviction**: All the cached samples are reused exactly r times before being evicted from the cache.
- **Uniqueness of Composed Augmentation**: Any composition of partial and final augmentation functions always produces a unique fully augmented sample.

$$\forall_{f_P, g_P \in A_P} \forall_{f_F, g_F \in A_F} \forall_{x \in \mathcal{X}} ((f_P \neq g_P) \text{ or } (f_F \neq g_F)) \to (f_F \circ f_P)(x) \neq (g_F \circ g_P)(x) \qquad A(x) = \{f_1(x), \ldots, f_{|A|}(x)\}$$

Data Refurbishing

- **Uniqueness of Composed Augmentation**: Any composition of partial and final augmentation functions always produces a uniq ue fully augmented sample.

$$\forall_{f_P,g_P \in A_P} \forall_{f_F,g_F \in A_F} \forall_{x \in \mathcal{X}}$$
$$((f_P \neq g_P) \; or \; (f_F \neq g_F)) \rightarrow (f_F \circ f_P)(x) \neq (g_F \circ g_P)(x)$$

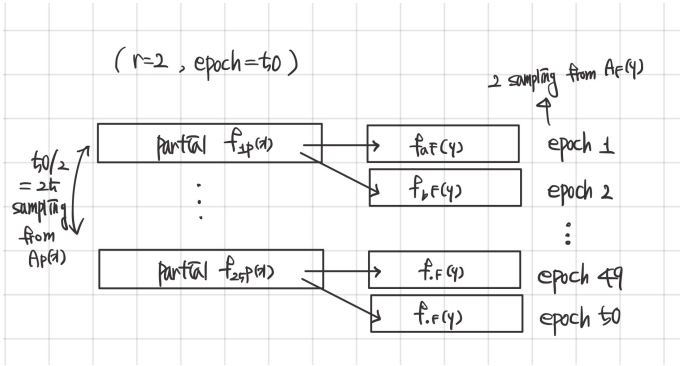
$$A := \{f_1, f_2, \ldots, f_{|A|}\} \text{ for } \forall_i \, f_i : \mathcal{X} \rightarrow \mathcal{X}'.$$
$$\{f_F \circ f_P | f_P \in A_P, f_F \in A_F\} = A.$$

- A(x): the set of all possible augmented samples produced by an augmentation A given an input sample.

$$A(x) = \{f_1(x), \ldots, f_{|A|}(x)\}$$
$$|A_P| \times |A_F| = |A| = |A(x)|$$

- Data refurbishing for k epochs with reuse factor r to an augmentation A for an input sample x

  = sampling process such that samples are taken r times from A_F(y) for every y sampled k/r times from A_P(x)



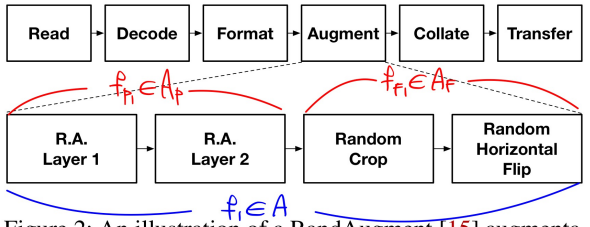


Figure 2: An illustration of a RandAugment [15] augmentation pipline in a typical data preparation pipeline. R.A. Layer stands for a RandAugment layer.

Data Refurbishing

- We can save computation without significant loss of the model generalization as long as the final augmentation provides sufficient sample diversity.

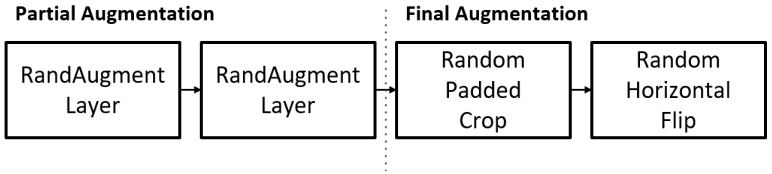
- Split policy goals:

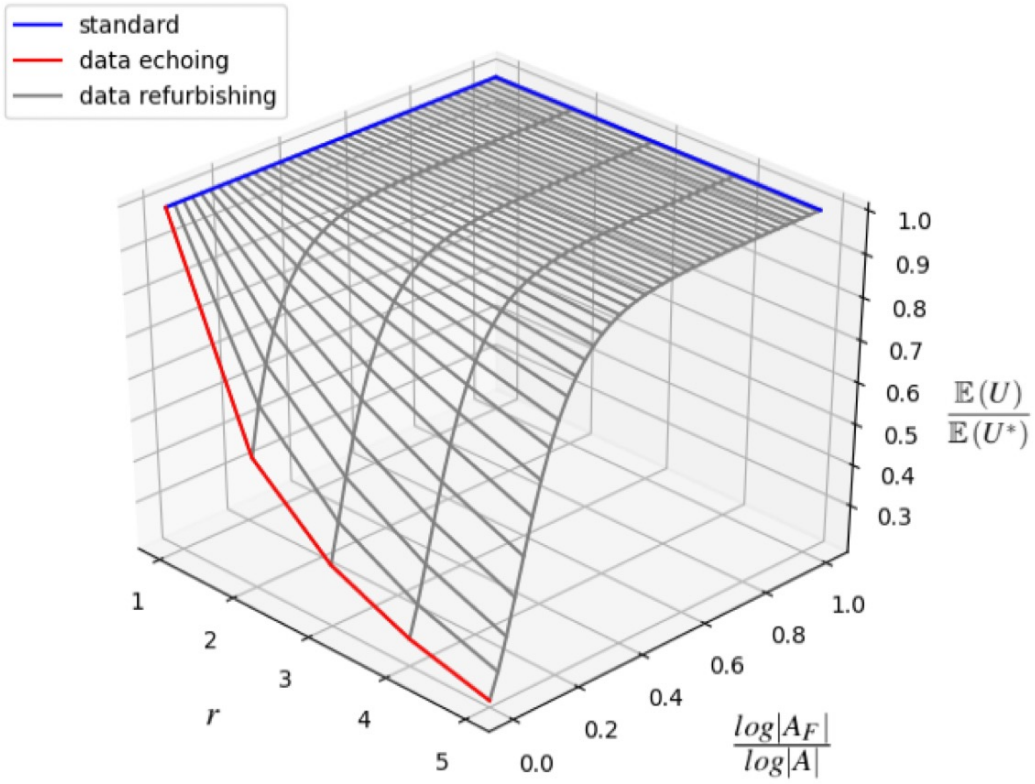
  1. **Final augmentation has enough diversity**.

  2. **Final augmentation has low computation overhead.**

- Example:

  RandAugment layer vs. Random Crop layer





Revamper

- Cache misses may fluctuate in the CPU processing time because non-cached samples require both partial and final augmentation whereas cached sample only need final augmentation.
- Overcome: keeping the # of cache misses constant both across epochs and within each epoch.

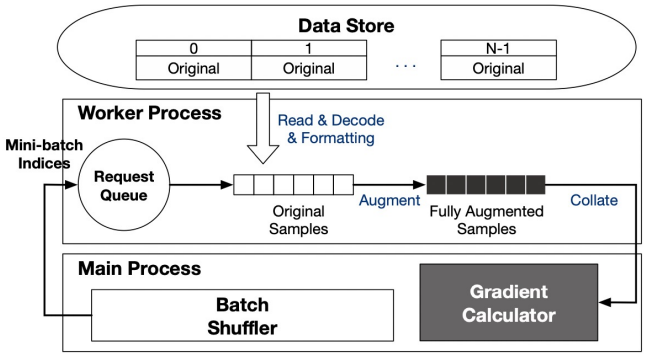


Figure 5: The architecture of a traditional data loading system (PyTorch dataloader).

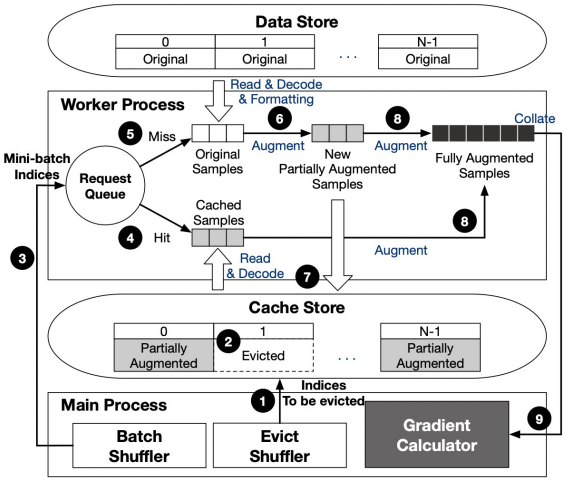


Figure 6: The architecture of Revamper and its end-to-end data preparation procedures.

- **The cache store**: added to store partially augmented samples.
- **The evict shuffler**: added in the main process, to select the indices to be evicted from the cache store accordin g to the balanced eviction.
- **The batch shuffler**: modified to sample mini-batch indices according to the cache-aware shuffle.

Revamper

Balanced Eviction

- Cache eviction policy to address inter-epoch computation skew
- **At the start of each training epoch, the evict shuffler samples N/r indices to be evicted.**
  (N: the # of training samples)
- The balanced eviction evenly distributes the computation overhead across epochs, by evicting the same amount of partially augmented samples in each epoch.

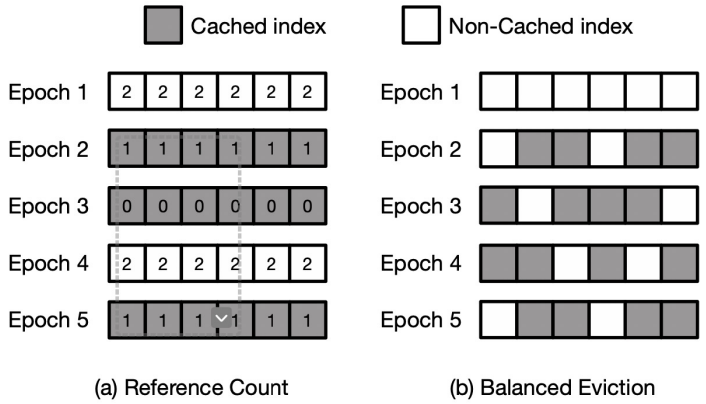


Figure 7: An example distribution of cache misses with (a) reference count algorithm and (b) the balanced eviction.

Cache Aware Shuffle

- To address intra-epoch computation skew
- **Each mini-batch has the same ratio of cached to non-cached samples.**
- Makes the processing time for all mini-batches stable.
- Ensure the randomness of the mini-batch indices by randomly sampling from both non-cached indices and cached indices.

Evaluation

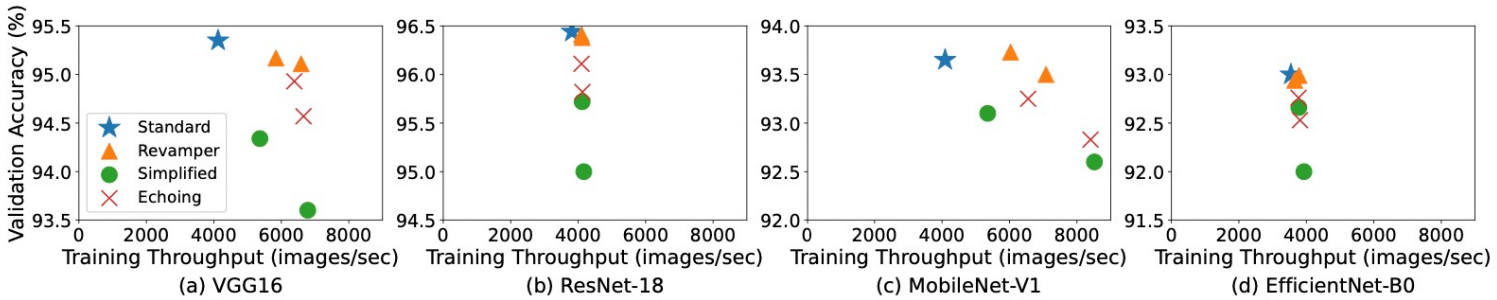


Figure 10: Training throughput and top-1 validation accuracy of DNN models trained on CIFAR-10 using RandAugment. Different points of the same setting represent measurements under different reuse factors (2 or 3) for Revamper and data echoing and under different numbers of removed transformation layers (1 or 2) for the simplified setting.

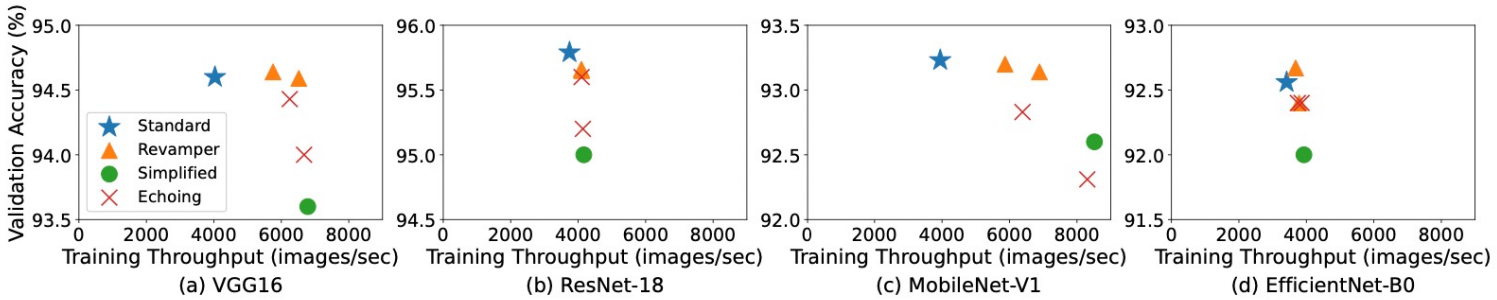


Figure 11: Training throughput and top-1 validation accuracy of DNN models trained on CIFAR-10 using AutoAugment. Different points of the same setting represent the results under different reuse factors (2 or 3).

Evaluation

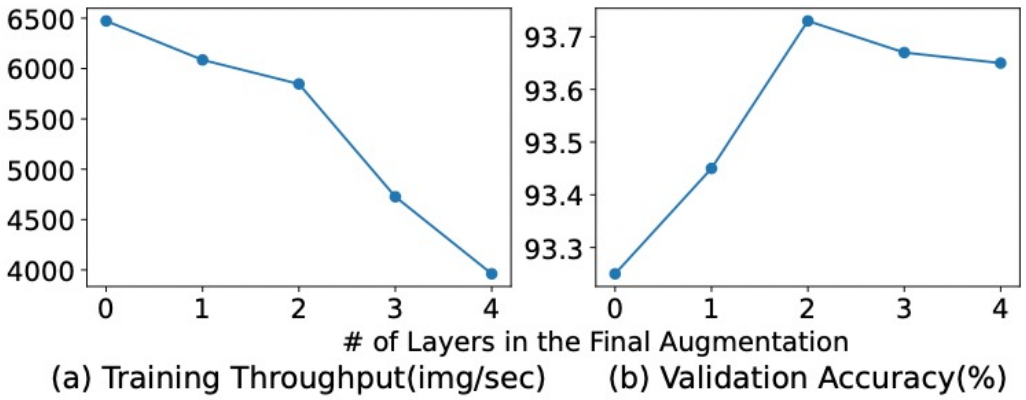


Figure 12: The training throughput and the top-1 validation accuracy for different split policies (MobileNet-V1 on CIFAR-10).

Reference

- Data Augmentation: https://hoya012.github.io/blog/Image-Data-Augmentation-Overview/
- Training Pipeline and basic concept of CPU overhead in data preprocessing: https://towardsdatascience.com/overcoming-data-preprocessing-bottlenecks-with-tensorflow-data-service-nvidia-dali-and-other-d6321917f851
- Choi, Dami, et al. "Faster neural network training with data echoing." arXiv preprint arXiv:1907.05550 (2019).
- Lee, Gyewon, et al. "Refurbish Your Training Data: Reusing Partially Augmented Samples for Faster Deep Neural Net work Training." 2021 USENIX Annual Technical Conference (USENIX ATC 21). 2021.

Thank You For Listening