Automatic Data Sampling Schedules via Population Based Training

Lei Cui*, Zhiyuan You*, Fenggang Liu, Yangguang Li, Bin Huang, Xin Lu, Ming Sun, Lifeng Sun, *Member, IEEE*

Abstract—An optimal data sampling schedule can efficiently boost neural networks performance. However, an effective sampling schedule is difficult to learn due to the inherently high dimension of parameters in learning the sampling schedule. In this paper, we propose an adaptive data sampling schedules method (AutoSampling) to automatically learn sampling schedules for model training, which consists of the multi-exploitation step aiming for optimal local sampling schedules and the exploration step for the ideal sampling distribution. More specifically, for exploitation, we achieve sampling schedule search with shortened exploitation cycle to provide enough supervision. For exploration, we periodically estimate the sampling distribution from the learned sampling schedules and perturb it to search in the distribution space. The combination of two searches allows us to learn a robust sampling schedule. The proposed AutoSampling can adaptively withdraw relatively important samples, which leads to overpassing similar and noisy samples. We not only perform extensive experiments on a variety of image classification tasks but also on different neural network architectures. AutoSampling can obtain robust improvements and outperform existing sampling methods.

Index Terms—Data Sampling Schedules, Hyper-parameter Learning, AutoML, Computer Vision.

1 Introduction

DATA sampling can act as a pivotal role in the training dynamics of neural networks. It has been widely used to reduce biases or noise in the dataset. For instance, data resampling, reweighting, and importance sampling, promote better model performance by adjusting the training data frequency and order [1], [2], [3], [4], [5], [6], [7]. It can even be used to reduce training time by removing "redundant" examples in the dataset [8]. Thus, finding an optimal dataset schedule is essential to effectively train neural networks. However, current methods rely on handcrafted rules that are based on assumptions over the dataset. Such heuristics prevent current methods to adapt well to new datasets. Learning-based methods [9], [10], [11] were proposed to automatically learn such sampling schedule. They did so by automatically reweighting or selecting training data utilizing meta-learning techniques or a policy network.

However, these methods still rely on human priors as proxies to optimize sampling policies, which may fail in practice. Such priors often include assumptions on policy network design for data selection [11], or dataset conditions like noisiness [9], [12] or imbalance [13]. These approaches take image features, losses, importance, or their representations as inputs and apply the policy network or other learning approaches with a small amount of parameters for estimating the sampling probability. However, relying on the aforementioned priors would, for instance, cause similar images to have similar sampling probability. This is due to images with similar visual features having a similar loss value when fed into the policy network. In turn, this would bias the training towards such samples. Therefore, we propose to directly optimize the sampling schedule itself so that no prior knowledge is required for the dataset. Specifically, the sampling schedule refers to the order by which data are selected for the entire training course. In this way, we only rely on data themselves to determine the optimal sampling schedule without any prior.

Directly optimizing a sampling schedule is challenging due to its inherent high dimension. For example, for the ImageNet classification dataset [14] with around one million samples, the dimension of parameters would be in the same order. While popular approaches such as deep reinforcement learning [15], [16], Bayesian optimization [17], population-based training [18] or simple random search [19] have already been utilized to tune low-dimensional hyper-parameters like augmentation schedules, their applications in data sampling schedules remain unexploited. As an example, in [15] the dimension of the data augmentation policy is only in dozens. Yet, such a method requires thousands of training runs to sample enough rewards to find an optimal augmentation policy. High-quality rewards require many epochs of training, thus hindering the training. As such, optimizing a sampling schedule may require orders of magnitude more rewards than data augmentation to gather and hence training runs, which results in prohibitively slow convergence. To overcome the aforementioned challenge, we propose a data sampling policy search framework, named AutoSampling, to sufficiently learn an optimal sampling schedule in a population-based training fashion [18]. Unlike previous methods, which focus on collecting long-term rewards and updating hyper-parameters or agents offline, our AutoSampling method collects rewards online with a shortened collection cycle but without priors. Specifically, the AutoSampling collects rewards within several training iterations, tens or hundred times shorter than that in existing works [15], [20]. In this manner, we provide the search process with much more frequent feedback to ensure sufficient optimization

- *L. Cui is with the Department of Computer Science at Tsinghua University. E-mail: cuil19@mails.tsinghua.edu.cn; * Equal contribution.*

of the sampling schedule. Each time when a few training iterations pass, we collect the rewards from the previous several iterations, accumulate them and later update the sampling distribution using the rewards. Then, we perturb the sampling distribution to search in distribution space and use it to generate new mini-batches for later iterations, which are recorded into the output sampling schedule. As illustrated in Sec. 5.4, shortened collection cycles with less interference can also better reflect the training value of each data.

Our contributions are as follows:

- To our best knowledge, we are the first to propose to directly learn a robust sampling schedule from the data themselves without any human prior or condition on the dataset.
- We propose the AutoSampling method to handle the optimization difficulty due to the high dimension of sampling schedules, and efficiently learn a robust sampling schedule through shortened reward collection cycle and online update of the sampling schedule.
- We conduct extensive experiments on many architectures and datasets to verify their robustness. Furthermore, we find that the learned sampling distributed by AutoSampling is determined by datasets distribution and is agnostic to architectures.

This paper will begin by examining related works in Section 2. The following Section 3 provides a brief overview of the method. We illustrate the proposed AutoSampling method in Section 4. Finally, the learned sampling schedule by AutoSampling is examined in Section 5.

It should be noted that this paper is an extension of our preliminary conference versions [21]. The present work adds extensive experiments which demonstrate the strong performance of our method on a 9 vision tasks, including CIFAR-10, CIFAR-100, ImageNet as well as fine-grained classification, such as Stanford Car. Different architectures are applied to show the robustness of our proposed AutoSampling method. Furthermore, we add considerable new experimental results in terms of ablation study to analyze the effectiveness of AutoSampling, including augmentation, dataset size, noisy dataset and so on.

2 BACKGROUND

2.1 Related Works

Data sampling is of great significance to deep learning and has been extensively studied. Approaches with human-designed rules take pre-defined heuristic rules to modify the frequency and order by which training data is presented. In particular, one intuitive method is to resample or reweight data according to their frequencies, difficulties, or importance in training [1], [2], [3], [4], [5], [6], [7], [12], [13], [22], [23], [24]. Reweighting approaches such as [25] aims to tackle the problem of long-tailed data distribution. [25] draws our attention to the loss function by providing a framework to reweight samples according to the number of effective samples. AutoSampling differs in several respects, it does not reweight samples and it can adapt to the training stage of the model. Another line of work mimics the way human learns. It follows a curriculum by sorting the data from easy to difficult according to a difficulty function [3], [10], [13]. However, [26] observed that curriculum learning has only marginal benefits on CIFAR100. In contrast, on CIFAR100, AutoSampling can increase the top-1 accuracy by up to 2.19%. These methods have been widely used in imbalanced training or hard mining problems. However, they are often restricted to certain tasks and datasets based on which they are proposed, and their ability to generalize to a broader range of tasks with different data distribution may be limited. In another word, these methods often implicitly assume certain conditions on the dataset, such as cleanness or imbalance. In addition, learning-based methods have been proposed for finding suitable sampling schemes automatically. For example, methods using meta-learning or reinforcement learning are taken to automatically select or reweight data during training [9], [10], [11], [27], but they are only tested on small-scale or noisy datasets. Whether or not they can generalize over tasks of other datasets still remain untested. In this work, we directly study the data sampling without any prior, and we also investigate its wide generalization ability across different datasets such as CIFAR-10, CIFAR-100, and ImageNet using many typical networks.

As for hyper-parameter tuning, popular approaches such as deep reinforcement learning [15], [16], Bayesian optimization [17] or simply random search [19] have already been utilized to tune low-dimensional hyper-parameters and proven to be effective. Nevertheless, they have not been adopted to find good sampling schedules due to their inherent high dimension. Some recent works tackle the challenge of optimizing high-dimensional hyper-parameter. [28] uses structured best-response functions and [29] achieves this goal through the combinations of the implicit function theorem and efficient inverse Hessian approximations. However, they have not been tested on the task of optimizing sampling schedules, which is the major focus of our work in this paper.

2.2 Population Based Training

Hyper-parameter tuning task can be framed as a bi-level optimization problem with the following objective function,

$$\begin{gathered}\min_{h\in\mathcal{H}}\mathcal{L}(\theta^*,h)\\ subject\ to\ \theta^*=\arg\min_{\theta\in\Theta}\mathcal{L}(\theta,h)\end{gathered} \tag{1}$$

where θ represents the model weight and $h = (h_1, h_2, \cdots, h_T)$ is the hyper-parameter schedule for T training intervals. Population based training (PBT) [18] solves the bi-level optimization problem by training a population $\mathcal{P}$ of child models in parallel with different hyper-parameter schedules initialized:

$$\mathcal{P} = \{(\theta_i, h_i, t)\}_{i=1}^{N_p} \tag{2}$$

where θ_i, h_i respectively represents the child model weight, the corresponding hyper-parameter schedule for the training interval t on worker i, and N_p is the number of workers. PBT proceeds in intervals, which usually consists of several epochs of training. During the interval, the population of models are trained in parallel to finish the lower-level optimization of weights θ_i.

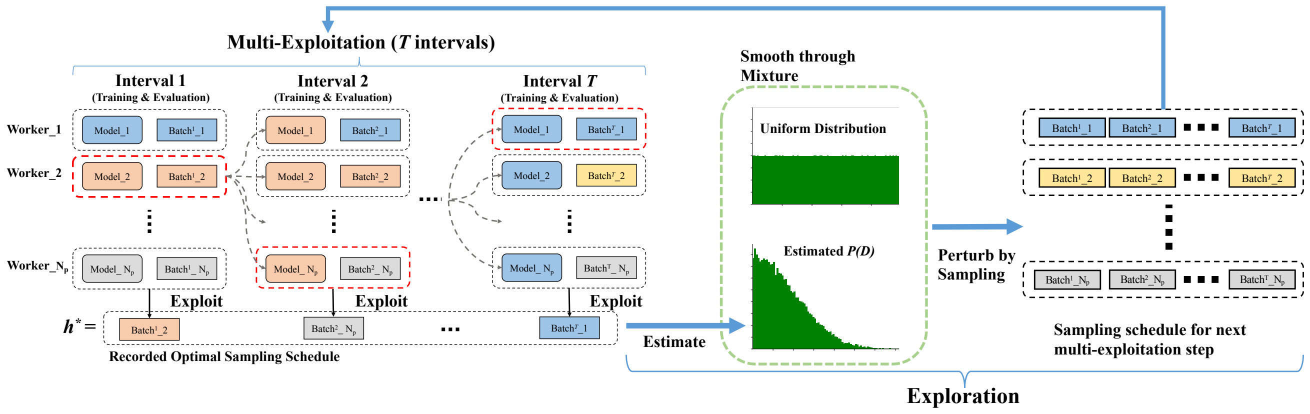

Multi-Exploitation (T intervals)
Interval 1 (Training & Evaluation)
Interval 2 (Training & Evaluation)
Interval T (Training & Evaluation)
Worker_1
Worker_2
Worker_N_p
Exploit
h* =
Recorded Optimal Sampling Schedule
Smooth through Mixture
Uniform Distribution
Estimated P(D)
Estimate
Perturb by Sampling
Sampling schedule for next multi-exploitation step
Exploration

Fig. 1. Overview of AutoSampling illustrated through one multi-exploitation-and-exploration cycle. a) The multi-exploitation step, illustrated by the left half, is the process of learning optimal sampling schedule locally. The same color of model for each worker indicates that the same model weight is cloned into it. Also for simplicity, in this figure, we adopt the exploitation interval of length 1. b) The exploration step, shown by the right half, is to search in the sampling distribution space. Specifically, we estimate the sampling distribution from the schedules collected in the multi-exploitation step and perturb it to generate new sampling schedules for all workers.

Between intervals, an exploit-and-explore procedure is adopted to conduct the upper-level optimization of the hyper-parameter schedule. In particular for interval t, to exploit child models are evaluated on a held-out validation dataset:

$$h_t^*, \theta_t^* = \underset{p_i=(\theta_i,h_i,t)\in\mathcal{P}}{\arg\max} \quad \text{eval}(\theta_i, h_i)$$
$$\theta^* \rightarrow \theta_i, i = 1, \cdots, N_p \tag{3}$$

The best performing hyper-parameter setting h_t^* is recorded and the top-performing model θ_t^* is broadcasted to all workers. To explore, new hyper-parameter schedules are initialized for interval $t+1$ with different random seeds on all workers, which can be viewed as a search in the hyper-parameter space. The next exploit-and-explore cycle will then be continued. In the end, the top-performing hyper-parameter schedule $h^* = (h_1^*, h_2^*, \cdots, h_T^*)$ can be obtained.

PBT is applied to tune low-dimenisal hyper-parameters such as data augmentation schedules [18], [20]. However, it cannot be directly used for finding sampling strategies due to the high dimension. For instance, in PBA [20] 3200 epochs of training are needed to optimize 60 hyper-parameters for data augmentation, and a linear up-scaling to learning sampling schedule in CIFAR-100 would require prohibitively 2.67 million epochs. Unlike PBT, our AutoSampling adopts a multi-exploitation-and-exploration structure, leading to much shorter reward collection cycles that contribute to much more effective rewards for sufficient optimization within a practical computational budget.

3 Preliminaries

Consider the bi-level optimization problem detailed by Equation.1 over a training dataset D. We define a sampling schedule to be an enumerated collection of data from dataset D for training the model, that is, $h = (x_1, \cdots, x_N) \in D^N$, with each x_i sampled from D. We use N to denote the total number of data trained for the lower task of optimizing the model weight θ. The product of N copies of dataset D constructs the search space $\mathcal{H} = D^N$ of sampling schedules, from which we wish to find an optimal sampling schedule to solve the aforementioned bi-level optimization problem:

$$\min_{h\in\mathcal{H}=D^N} \mathcal{L}(\theta^*, h)$$
$$subject\ to\ \theta^* = \underset{\theta\in\Theta}{\arg\min}\, \mathcal{L}(\theta, h) \tag{4}$$

Note a sampling schedule may also be represented as a enumerated collection of several sampling sub-schedules. For instance, a sampling schedule h can be denoted as $h = \{h_i | i = 1, \cdots, k\}$ where h is the concatenation of $(h_1, \cdots, h_k)$

4 AutoSampling with Searching

The overview of our AutoSampling is illustrated in Fig.1. AutoSampling alternately runs multi-exploitation step and exploration step. In the exploration step, we 1) update the sampling distribution using the rewards collected from the multi-exploitation step (the sampling distribution is uniform distribution initially); 2) perturb the updated sampling distribution for child models so that different child models have different sampling distributions; 3) use the corresponding perturbed sampling distribution for each child model to sample mini-batches of training data. In the multi-exploitation step, we 1) train multiple child models using the mini-batches sampled from the exploration step; 2) collect short-term rewards from the child models. AutoSampling finishes with a recorded top-performing sampling schedule, which can be transferred to other models.

4.1 Multi-Exploitation by Searching in the Data Space

In the multi-exploitation step, we aim to search locally in the data space by collecting short-term rewards and sub-schedules. Specifically, we wish to learn a sampling schedule for T exploitation intervals. In each interval, there are a population $\mathcal{P}$ of N_p child models. We denote $\mathbf{h}_{t,i}$ as the training data sub-schedule in the t^{th} interval for the i^{th} child model. When all of the T exploitation intervals for the i^{th} child model are considered, we have $\mathbf{H}_i = \{\mathbf{h}_{t,i} | t = 1, \ldots, T\} = (x_1, \cdots, x_{N_T})$, where N_T is

Algorithm 1 The Multi-Exploitation Step

Input: Training dataset D, population $\mathcal{P} = \{(\theta_i, h_i, t)\}_{i=1}^{N_p}$, population size N_p, number of exploitation intervals T, exploitation interval length N_s

```
Initialize $\mathbf{H}^* \leftarrow ()$
for $t$=1 to $T$ do
    for $j$=1 to $N_s$ do
        for $(\theta_i, h_{t,i}, t) \in \mathcal{P}$ do
            $\theta_i \leftarrow \bigtriangledown\mathcal{L}(\theta_i, h_{t,i})$   ▷ update the weight of child model i
        end for
        $h_t^*, \theta_t^* = \arg\max_{\mathcal{P}} \text{eval}(\theta_i, h_i)$
        $\mathbf{H}^* \leftarrow \mathbf{H}^* + h_t^*$   ▷ update the sampling for child model i
        for $i = 1$ to $N_p$ do
            $\theta_i \leftarrow \theta_t^*$   ▷ clone the optimal weight
        end for
    end for
end for
Return $\mathbf{H}^*$, $\mathcal{P}$
```

Algorithm 2 Search based AutoSampling

Input: Training dataset D, population size N_p

```
Initialize $H^* \leftarrow ()$ , $P(D) \leftarrow$ uniform($D$) and initialize child models $\theta_1, \cdots, \theta_{N_p}$
while not end of training do
    for $i = 1$ to $N_p$ do
        Sample $h_i$ from Mixture(log($P(D) + \beta$), $N_u \times$ uniform($D$))
    end for
    Initialize $\mathcal{P} = \{(\theta_i, h_i, t)\}_{i=1}^{N_p}$
    $\mathbf{H}^*, \mathcal{P} \leftarrow$ Alg.1
    Estimate $P(D)$ according to Equation (6)
    Update $P(D)$ according to Equation (7)
    $H^* \leftarrow H^* + \mathbf{H}^*$
end while
Return $H^*$, $P(D)$
```

the number of training data used for this multi-exploitation step. Each interval consists of N_s training iterations that is also equivalent to N_s training mini-batches, where N_s is the length of the interval.

AutoSampling is expected to produce a sequence of training samples, denoted by $\mathbf{H}^*$, so that a given model is optimally trained.

The population $\{\mathbf{H}_i\}$ forms the local search space, from which we aim to search for an optimal sampling schedule $\mathbf{H}^*$.

Given the population $\mathcal{P}$, we train them in parallel on N_p workers for interval t. Once the interval of data $\mathbf{h}_{t,i}$ containing N_s training batches have been used for training, we evaluate all child models and use the top evaluation performance as the reward. According to the reward, we record the top-performing weight and sub-schedule for the current interval t, in particular,

$$h_t^*, \theta_t^* = \underset{p_i=(\theta_i, h_i, t)\in\mathcal{P}}{\arg\max} \text{eval}(\theta_i, h_{t,i}) \tag{5}$$

On the other hand, we update all child model weights of $\mathcal{P}$ by cloning into them with the top-performing weight θ_t^* so we can continue searching based on the most promising child. We continue the exploit steps through all exploitation intervals, record and output the recorded optimal sampling schedule $\mathbf{H}^* = \{h_1^*, h_2^*, \cdots, h_T^*\}$ for the multi-exploitation step. By using exploitation interval of mini-batches rather than epochs or even entire training runs adopted by earlier methods, AutoSampling may yield a better and more robust sampling schedule. It should be pointed out that even though in AutoSampling rewards are collected within a much shorter interval, they remain effective. As we directly optimize the sampling schedule, we are concerned with only the data themselves. The short-term rewards reflect the training value of data from the exploitation interval they are collected. But for global hyper-parameters such as augmentation schedules, short-term rewards may lead to inferior performance as these hyper-parameters are concerned with the overall training outcome. We describe the multi-exploitation step with details in Alg.1.

4.2 Exploration by Searching in Sampling Distribution Space

In the exploration, we search in the sampling distribution space by updating and perturbing the sampling distribution. We first estimate the underlying sampling distribution $P(D)$ from the top sampling schedule h^* produced in the multi-exploitation, that is, for $x \in D$,

$$P(x) = \frac{count(x \in \mathbf{H}^*)}{\sum_{x\in D} count(x \in \mathbf{H}^*)} \tag{6}$$

where $count(x \in \mathbf{H}^*)$ denotes the number of x's appearances in $\mathbf{H}^*$. We further perturb the $P(D)$ and generate the sampling schedules on each worker for the later multi-exploitation. We introduce perturbations into the generated schedules by simply sampling from the multinomial distribution $P(D)$ using different random seeds. However, in our experiments, we observe that the distribution produced by $P(D)$ tends to be extremely skewed and a majority of the data actually have zero frequencies. Such skewness causes highly imbalanced training mini-batches, and therefore destabilizes subsequent model training.

Distribution Smoothing To tackle the above issue, we first smooth $P(D)$ through the logarithmic function, and then apply a probability mixture with uniform distributions. In particular for the dataset D,

$$P'(D) = Mixture(\log(P(D) + \beta), N_u \times \text{uniform}(D)) \tag{7}$$

where $\beta \geq 1$ is the smoothing factor and $N_u \times \text{uniform}(D)$ denotes N_u uniform multinomial distributions on the dataset D. The smoothing through the $\log$ function can greatly reduce the skewness, however, $\log(P(D) + \beta)$ may still contain zero probabilities for some training data, resulting in unstable training. Therefore, we further smooth it through a probability mixture with N_u uniform distribution uniform(D) to ensure presence of all data. This is equivalent to combining N_u epochs of training data to the training batches sampled from $P(D)$, and shuffling the union. Once we have new diverse sampling schedules for the population, we proceed to the next multi-exploitation step.

We continue this alternation between multi-exploitation and exploration steps until the end of training. Note that to generate sampling schedule for the first multi-exploitation run, we initialize $P(D)$ to be an uniform multinomial distribution. In the end, we output a sequence of optimal sampling schedules $H^* = \{\mathbf{H}_1^*, \cdots, \mathbf{H}_n^*\}$ for n alternations. The entire process is illustrated in details in Alg.2.

5 Experiments

In this section, we present comprehensive experiments on various architectures and datasets to illustrate the effectiveness and robustness of AutoSampling. Extensive ablation experiments are constructed to explore why AutoSampling is preferable. The process of progressively learning sampling distribution is also shown to demonstrate the principle of AutoSampling.

5.1 Datasets and Architectures

Datasets. 9 image classification datasets are utilized. These datasets include general objects classification (CIFAR-10 [30], CIFAR-100 [30], ImageNet [14], Caltech-101 [31]); fine-grained object classification (Stanford Cars [32], FGVC Aircraft [33]); human related classification (FER2013 [34], UCF101 [35]); scene classification (SUN387 [36]). For the video datasets UCF101, we use the middle frame of each video sequence as the input image.
Architectures. Various network architectures are used to verify the generalization of AutoSampling, including a host of ConvNets(ResNet, DenseNet, MobileNet, and ShuffleNet) and a pure transformer-based model(Vit). ConvNets are constructed based on the convolution operator, while transformers are based on the self-attention operator.

5.2 Implementation Details

For all experiments, we use the standard inception-style augmentation [37], i.e., a random image crop and a horizontal flip with a probability 50%. All classification datasets use the basic settings, except especially demonstrated. For the basic settings, we adopted the learning rate of 0.1 and a step decay learning schedule where the learning rate is divided by 5 after every 60 epochs. We run the experiments with 240 epochs of training. In addition, we set the training batch size to be 64 per worker, and each worker is for one Nvidia V100 GPU card. 20 workers are used as default. The input images are resized to 224×224 while training and 256×256 while testing. What's more, we run the explore step for each $N_u + 1$ epochs with $N_u = 3$, but note that we take the first explore step after the initial 20 epochs to better accumulate enough rewards.
Experiments on CIFAR. We use the same training configuration for both CIFAR-100 and CIFAR-10 datasets, which both consist of 50000 training images. We use most basic settings except that the training batch size is set to 128 per worker, the experiments require 4800 epochs of training for 20 workers, and roughly 14 hours of training time, and the input images are resized to 32×32.
Experiments on ImageNet. For ImageNet which consists of 1.28 million training images, we adopted the base learning rate of 0.2 and a cosine decay learning rate schedule. We run the experiments with 100 epochs of training. For each worker, we utilize eight Nvidia V100 GPU cards and a total batch size of 512. Eight workers are used for all ImageNet experiments, and the rest of the setting adheres to that of CIFAR experiments. In addition, we utilize FP16 computation to achieve faster training, which has almost no drop in accuracy in practice. The experiments require 800 epochs of training for 8 workers and roughly 4 days of training time.
Experiments on Vision Transformers. We run all experiments on ViT [38] on CIFAR100. Due to the optimization difficulty of Vision Transformers [39], we use a pre-trained ViT-S [39] and fine-tune it for 100 epochs on CIFAR100 following [39]. The input images are resized to 224×224 for both training and testing. The rest of the setting is the same as that of the basic settings. The experiments require 2000 epochs of training for 20 workers and roughly 10 hours of training time.

5.3 Results

Experiment on various classification datasets. We report the results on 9 classification datasets in Table 1. Several different convolution networks are also used to verify the effectiveness of AutoSampling. The results indicate that compared with the regular mini-batches uniformly sampling scheduler, AutoSampling can obtain robust improvements, no matter on general objects classification tasks, fine-grained object classification tasks, human-related classification tasks, or scene classification tasks. The mixture exploration outperforms the baseline by a clear margin. Furthermore, We notice that there is a different improvement on diverse datasets, e.g., ResNet18 trained on UCF101 datasets can be improved up to 3.18% while on CALTECH datasets only 0.3%. One possible explanation is the difficulty of the dataset, which may be influenced by class number, data size, the quality of images, and noise.
Comparison with existing sampling methods. To better illustrate the effectiveness of our AutoSampling method, we conduct experiments in comparison with recent non-uniform sampling methods DLIS [4] and RAIS [5]. DLIS [4] achieves faster convergence by selecting data reducing gradient norm variance, while RAIS [5] does so through approximating the ideal sampling distribution using robust optimization. The comparison is recorded in Table 2.

First, we run AutoSampling using Wide Resnet-28-2 [40] on CIFAR-100 with the training setting aligned roughly to [5]. AutoSampling achievs improvement of roughly 3 percentage points (73.37±1.09% $\rightarrow$ 76.24±1.02%), while [5] shows improvement of 2 percentage points (66.0% $\rightarrow$ 68.0 %). Second, we report the comparisons between AutoSampling and RAIS on CIFAR-100. [4] shows no improvement (76.4% $\rightarrow$ 76.4 %) on accuracy and 0.027 (0.989 $\rightarrow$ 0.962) decrease in validation loss, while our method shows improvement of 0.008 (78.6% $\rightarrow$ 79.4%) on accuracy and 0.014 (0.886 $\rightarrow$ 0.872) decrease in validation loss. As such, our method demonstrates improvements over existing non-uniform sampling methods.

5.4 Ablation study

For this part, we gradually build up and test components of AutoSampling on CIFAR-100, and then examine their

TABLE 1
Performance on classification datasets using AutoSampling and baselines. Uniform represents the regular mini-batches uniformly sampled from the dataset. Mixture represents our AutoSampling method.

NETWORK	DATASETS	CLASS	DATA_SIZE	UNIFORM (%)	MIXTURE (%)	IMPROVEMENT (%)
RESNET18	CALTECH	102	3.1K	69.79	70.09	+0.3
	AIRCRAFT	100	6.7K	78.97	80.71	+1.75
	STANFORDCAR	196	8.1K	85.41	88.20	+2.79
	UCF101	101	9.5K	45.02	48.2	+3.18
	SUN	397	19.9K	48.22	49.76	+1.54
	FER2013	7	28.7K	70.72	71.74	+1.02
	CIFAR10	10	50K	93.01	95.80	+1.79
	CIFAR100	100	50K	78.46	79.44	+0.98
	IMAGENET	1000	1281K	70.38	72.91	+2.53
RESNET50	STANFORDCAR	196	8.1K	87.19	88.99	+1.8
	CIFAR10	10	50K	93.60	96.09	+2.49
	CIFAR100	100	50K	79.70	81.53	+1.83
MOBILENETV2	STANFORDCAR	196	8.1K	87.33	90.39	+3.03
	CIFAR100	100	50K	68.91	70.11	+1.2
SHUFFLENETV2	STANFORDCAR	196	8.1K	87.33	90.25	+2.92
	CIFAR100	100	50K	71.65	73.04	+1.39

TABLE 2
Comparisons among AutoSampling and existing sampling methods on CIFAR-100. Baseline represents the regular mini-batches uniformly sampling schedule.

Methods	Network	Baseline (%)	With method (%)	Improvement (%)
DLIS	WRN-28-2	66.0	68.0	+2.0
AutoSampling (ours)	WRN-28-2	73.37±1.09	76.24±1.02	**+2.87**
RAIS	ResNet18	76.4	76.4	+0.0
AutoSampling (ours)	ResNet18	78.46±0.035	79.44±0.020	**+0.98**

performances on CIFAR-10 and ImageNet datasets. Except especially demonstrated, all experiments used the basic settings.

Exploration types We first introduce the three exploration types examined, corresponding to one baseline and two variants of AutoSampling.

- **Uniform Exploration** corresponds to the regular model training with mini-batches uniformly sampled from the dataset.
- **Random Exploration** adds the multi-exploitation step upon the uniform exploration. In particular, the random exploration method conducts the multi-exploitation step (4.1) among several workers. In between multi-exploitation steps, the random exploration generates later sampling schedulers for each worker simply through uniform sampling. Note the random exploration with one worker is equivalent to the uniform exploration.
- **Mixture Exploration** adds the sampling distribution search (4.2) upon the random exploration in between multi-exploitation steps, completing the AutoSampling method.

Adding Exploration Type. We further add mixture as the exploration type to see the effects of learning the underlying sampling distribution, and completing the proposed method. As shown in Table 3, with ResNet-18 and ResNet-50 we push performance higher with the mixture exploration, and outperform the baseline method by about 1 and 1.8 percentage on CIFAR-100 respectively. However, we found that it is not true in the case of DenseNet-121 and this case may be attributed to the bigger capacity of DenseNet-121.

Adding Workers To look into the influence of the worker numbers, we conduct experiments using worker numbers of 1, 20, 80 respectively with the same setting ($N_s = 20$ with random exploration). With the worker number of 1, the experiment is simply the normal model training using stochastic gradient descent. To show the competitiveness of our baselines, we also include recent state-of-the-art results on CIFAR-100 with ResNet-18 and ResNet-50 [41], [42]. We notice significant performance gain using the worker number of 20 for ResNet-18, ResNet-50 and DenseNet-121 [43], [44], as illustrated in Table 3. However, we note that increasing worker number from 20 to 80 only brings marginal performance gains across various model structures, as shown in Table 3. Therefore, we set the worker number to be 20 for the rest of the experiments.

Shortening Exploitation Intervals. To study the effects of the shortened exploitation interval, we run experiments using different exploitation intervals of 20 and 80 batches(iterations) respectively. As shown in Table 3, models with the shorter exploitation interval of 20 batches(iterations) perform better than the one with the longer exploitation interval across all three network structures, conforming to our assumptions that the reward collected reflects value of each data used in the

TABLE 3
Ablation study experiments of AutoSampling. Worker(N_p) is the number of workers used, equivalent to the population size. Interval(N_s) is the exploitation interval in terms of batches or equivalent training iterations.

Dataset	Network	Worker(N_p)	Interval(N_s)	Exploration type	top1(%)
CIFAR-100	ResNet18 [41]	-	-	-	78.34±0.05
	ResNet18	1	-	Uniform	78.46±0.035
	ResNet18	20	80 Batches	Random	78.76±0.003
	ResNet18	20	20 Batches	Random	78.99±0.003
	ResNet18	80	20 Batches	Random	79.09±0.017
	ResNet18	20	20 Batches	Mixture	**79.44**±0.020
	ResNet50 [42]	-	-	-	79.34
	ResNet50	1	-	Uniform	79.70±0.023
	ResNet50	20	80 Batches	Random	80.55±0.129
	ResNet50	20	20 Batches	Random	81.05±0.064
	ResNet50	80	20 Batches	Random	81.19±0.072
	ResNet50	20	20 Batches	Mixture	**81.53**±0.088
	DenseNet121	1	-	Uniform	80.13±0.028
	DenseNet121	20	80 Batches	Random	80.62±0.694
	DenseNet121	20	20 Batches	Random	**81.11**±0.127
	DenseNet121	80	20 Batches	Random	81.08±0.021
	DenseNet121	20	20 Batches	Mixture	80.97±0.006
	ViT-S	1	-	Uniform	88.90±0.014
	ViT-S	20	20 Batches	Random	89.62±0.008
	ViT-S	20	20 Batches	Mixture	**89.69**±0.009
	MobileNetV2	1	-	Uniform	68.91±0.014
	MobileNetV2	20	20 Batches	Random	**70.52**±0.008
	MobileNetV2	20	20 Batches	Mixture	70.11±0.009
	ShuffleNetV2	1	-	Uniform	71.65±0.034
	ShuffleNetV2	20	20 Batches	Random	72.74±0.028
	ShuffleNetV2	20	20 Batches	Mixture	**73.04**±0.029
CIFAR-10	ResNet18	1	-	Uniform	93.01±0.009
	ResNet18	20	20 Batches	Random	**95.86**±0.003
	ResNet18	20	20 Batches	Mixture	95.80±0.018
	ResNet50	1	-	Uniform	93.60±0.004
	ResNet50	20	20 Batches	Random	**96.10**±0.002
	ResNet50	20	20 Batches	Mixture	96.09±0.070
STANFORD CARS	ResNet18	1	-	Uniform	87.19±0.018
	ResNet18	20	20 Batches	Random	89.11±0.073
	ResNet18	20	20 Batches	Mixture	88.99±0.068
	MOBILENETV2	1	-	Uniform	87.33±0.013
	MOBILENETV2	20	20 Batches	Random	90.25±0.030
	MOBILENETV2	20	20 Batches	Mixture	**90.39**±0.048
	SHUFFLENETV2	1	-	Uniform	87.33±0.015
	SHUFFLENETV2	20	20 Batches	Random	89.42±0.044
	SHUFFLENETV2	20	20 Batches	Mixture	**90.25**±0.045
IMAGENET	ResNet18	1	-	Uniform	70.38
	ResNet18	20	20 Batches	Random	72.07
	ResNet18	20	20 Batches	Mixture	**72.91**
	ResNet34	1	-	Uniform	74.09
	ResNet34	20	20 Batches	Random	76.11
	ResNet34	20	20 Batches	Mixture	**76.92**

exploitation interval. This result adheres to our intuition that shorter exploitation interval can encourage the sampler to accumulate more rewards to learn better sampling schedules. For the rest of this section we keep the exploitation interval of 20.

5.5 Robustness

In this part, more experiments concerned about the robustness of Autosampling are constructed. Multiple factors, from data to model, are considered. All experiments are constructed on Aircraft datasets and based on the ResNet18 model. The other settings are the same as the basic settings.

Add pre-train model. The pre-train model is an indispensable method for computer vision tasks, which can improve models' performance steadily. With the pre-train model, models can converge more quickly and require less data and training time. To verify that our data sample scheduler is agnostic with the pre-train model, we apply AutoSampling to both training from scratch and training from the pre-train model. With the same training setting on Aircraft datasets, AutoSampling can also improve the performance of the pre-train model by 1.5%, as Table 4. Compared with training from scratch, the improvement of the pretrain model only decays 0.22%.

TABLE 4
Performance on pre-train model using AutoSampling and baselines.

METHOD	UNIFORM(%)	MIXTURE(%)	GAIN(%)
W/O PRE-TRAIN	78.97	80.71	+1.75
W/ PRE-TRAIN	81.55	83.08	+1.53

Stronger augmentation. Data augmentation and sample scheduler are both important ways to exploit the potential of data. In previous experiments, we all use the standard inception-style augmentation [37], i.e., a random image crop and a horizontal flip with a probability of 50%. In this part, we use stronger augmentation like AutoAugment [15] to validate the robustness of AutoSampling. The results(see Table 5) show that our sample scheduler works well with stronger augmentation. Training with AutoAugment, AutoSampling can further improve the classification performance by 1.682%. Compared with training strategies using uniform sampling and inception-style augmentation, AutoSampling and AutoAugment can increase the top1 accuracy by 6.09% in total.

TABLE 5
Performance on stronger augmentation using AutoSampling and baselines.

AUG	UNIFORM(%)	MIXTURE(%)	GAIN(%)
DEFAULT	78.97	80.71	+1.75
AUTOAUG	83.378	85.06	+1.682

Network architectures. Different network architectures have been experimented including ConvNet and transformer network. ConvNet and transformer have diverse principles. ConvNets are constructed based on the convolution operator, while transformer-based on the self-attention operator. Many tricks have different effects for two completely different networks. Such, ResNet, MobileNetv2 [45], DENSENET, and ShuffleNetv2 [46] as the representative of the Convolution network and VIT as the representative of the transformer network are chosen to verify the robustness of AutoSampling.

As shown in Table 3, AutoSampling can benefit both the Convolution network and transformer. Compared with a uniform sampling schedule, VIT can be improved by 0.79% while RESNET50 can be improved by 1.83%. The results indicate that the AutoSampling method can likely improve data usage efficiency, regardless of the model architecture.

Model size. To exclude the effect of model size, we conducted experiments on the series ResNet model including ResNet18R̃erNet152 whose parameters range from 10M to 100M. The results(see Table 6) indicate AutoSampling has a robust improvement on different scale models. No matter ResNet18 or RerNet152, AutoSampling can improve classification accuracy by at least 1%. Furthermore, we can also observe that AutoSampling may benefit more on small models.

TABLE 6
Performance on different model size using AutoSampling and baselines.

MODEL	PARAM	UNIFORM(%)	MIXTURE(%)	GAIN(%)
R18	11.7M	78.97	80.71	+1.75
R34	21.8M	80.68	82.24	+1.56
R50	25.6M	81.22	82.92	+1.70
R101	44.5M	83.08	84.19	+1.11
R152	60.2M	83.29	84.50	+1.21

Dataset size. As the Table 1, different dataset have different improvement. Different datasets have levels of difficulty, which is affected by dataset size, class number, and data quality. In this part, we concentrate on the dataset size factor. We use the Aircaft dataset for example and extract 10%, 30%, 60% of training data separately. All the experiments follow the basic settings, except that the batch size is set to 16 when using 30% data and is set to 8 when using 10% data. The results are shown in Table 7. It shows that dataset size indeed influences the improvement of AutoSampling. But adjusting the batch size can improve the effect.

TABLE 7
Performance on different size of Aircaft using AutoSampling and baselines.

SIZE	UNIFORM(%)	MIXTURE(%)	GAIN(%)
10%	7.921	12.21	+4.289
30%	43.86	44.16	+0.3
60%	60.34	61.6	+1.26
100%	78.97	80.71	+1.75

Noisy dataset. Another important factor affecting data quality is noise. It's intuitive that AutoSampling can suppress the sampling probability of noisy data to mitigate noise problems. To verify this hypothesis, we shuffle 5% and 10% of Aircaft's labels. All experiments with noise data use 16 batch size. The results are shown in Table 8. We can observe that AutoSampling improves performance robustly on noisy datasets. Furthermore, as the noise data increases, AutoSampling works better.

TABLE 8
Performance on noisy Aircaft dataset using AutoSampling and baselines.

NOISE	UNIFORM(%)	MIXTURE(%)	GAIN(%)
0%	78.97	80.71	+1.75
5%	74.59	76.60	+2.01
10%	69.127	71.56	+2.433

5.6 Static vs Dynamic Schedules

We aim to see if the final sampling distribution estimated by our AutoSampling is sufficient to produce robust sampling schedules. In another word, we wish to know training with the AutoSampling is either a process of learning a robust sampling distribution, or a process of dynamically adjusting

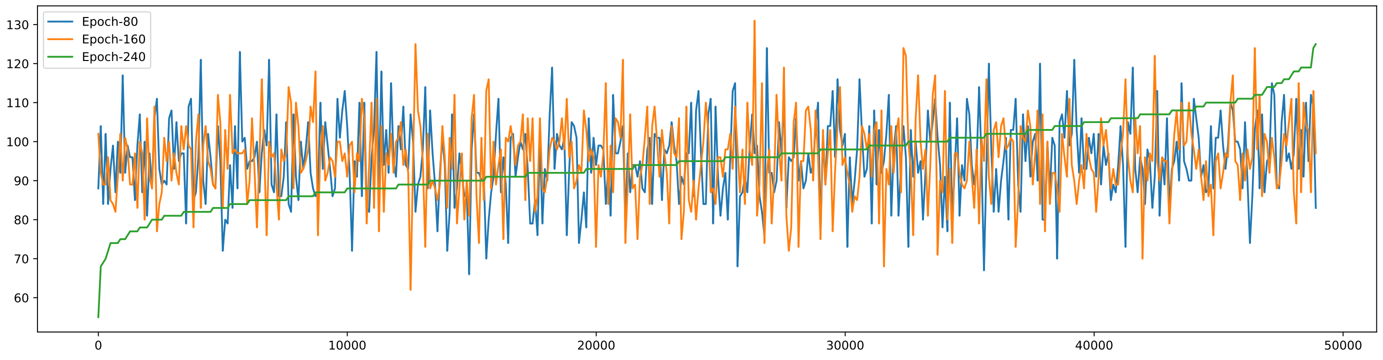


Fig. 2. The comparison between histograms estimated from the sampling schedules of Epoch 80, 160 and 240 from CIFAR-100 with ResNet-18. We divide the 50000 training images into 500 segments of 100 images, and calculate the histograms of total data counts of all segments. We reorder the x-axis based on the ranking of data counts for epoch 240 for easier comparison.

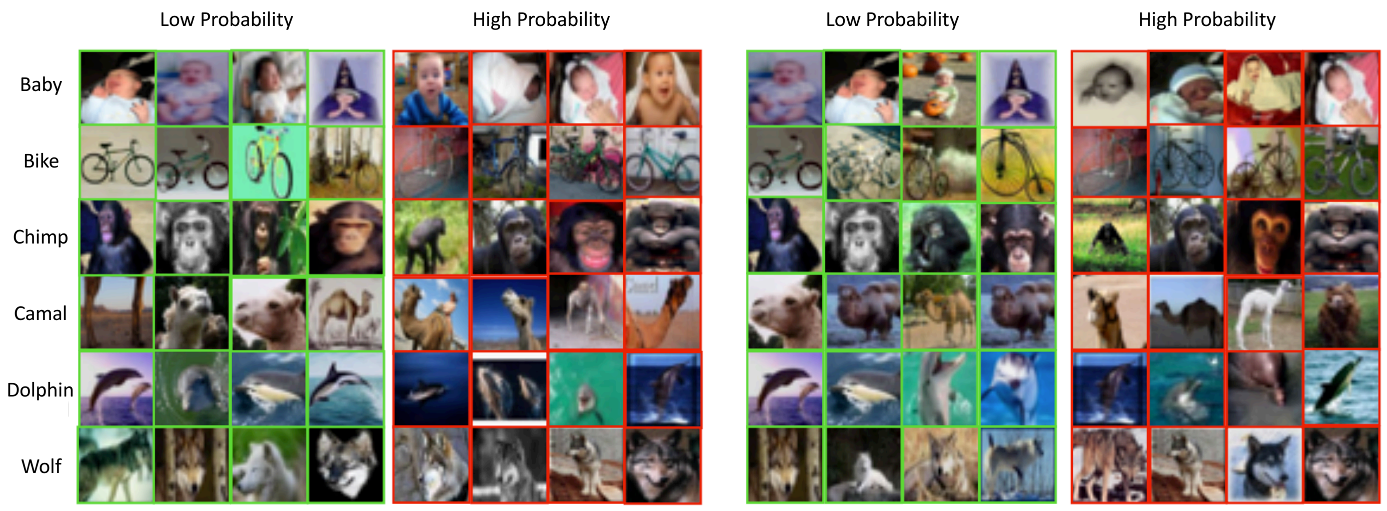


Fig. 3. Example images on the head and tail of the sampling spectrum. The images in green are the ones with low sampling probability, while the images in red are likely to be sampled. We obtain these images using AutoSampling with ResNet-50 (on the left) and ShuffleNetV2 (on the right) on CIFAR-100.

the sampling schedule for optimal training. To this end, we conduct training using different sampling schedules. First, we calculate the sampling distribution estimated throughout the learning steps of AutoSampling, and use it to generate the sampling schedule of a full training process, which we denote as STATIC. Moreover, we represent the sampling schedule learned using AutoSampling as DYNAMIC, since AutoSampling dynamically adjust the sampling schedule alongside the training process. Finally, we denote the baseline method as UNIFORM, which uses the sampling schedule generated from uniform distribution.

We report results on CIFAR-100 with ResNet-18 and ResNet-50 in Table 9. Model trained with STATIC sampling schedules exceeds the baseline UNIFORM significantly, indicating the superiority of the learned sampling distribution over the uniform distribution. It shows the ability of AutoSampling to learn good sampling distribution. Nonetheless, note that models trained with DYNAMIC sampling schedules outperform models trained with STATIC, by a margin bigger than the one between STATIC and UNIFORM. This result shows the fact that despite the AutoSampling's capability of learning good sampling distribution, its flexibility during training matters even more. Moreover, this phenomenon also indicates that models at different stages of learning process may require different sampling distributions to achieve optimal training. One single sampling distribution, even

TABLE 9
Static vs Dynamic sampling schedule on CIFAR-100 (%)

NETWORK	SAMPLING TYPE		
	UNIFORM	STATIC	DYNAMIC
RESNET18	78.46±0.035	78.80±0.007	**79.44**±0.020
RESNET50	79.70±0.023	80.21±0.014	**81.53**±0.088

TABLE 10
Transfer of sampling distributions learned by three model structures to ResNet-50 on CIFAR-100 (%). UNIFORM denotes the baseline result using uniform sampling distribution.

NETWORK	SAMPLING SCHEDULE SOURCE				
	UNIFORM	RESNET18	RESNET50	DENSENET121	VIT-S
RESNET50	79.70±0.023	80.27±0.014	80.21±0.014	80.47±0.194	80.91±0.021
VIT-S	88.90±0.011	88.98±0.010	88.95±0.009	-	-

gradually estimated using AutoSampling, seems incapable of covering the needs from different learning stages. We plot the histograms of data counts in training estimated from schedules of different learning stages with ResNet-18 on CIFAR-100 in Fig.2, showing the great differences between optimized sampling distributions from different epochs.

5.7 Analyzing sampling schedules learned by AutoSampling

To further investigate the sampling schedule learned by AutoSampling, we review the images at the tail and head part of the sampling spectrum. In particular, given a sampling schedule learned we rank all images based on their numbers of appearances in the training process. Training images at the top and bottom of the order are extracted, corresponding to high and low probabilities of being sampled respectively. In Fig.3, we show 6 classes of exemplary images. Conforming to our presumption, the sampling probability seems to indicate the difficulty of each training image. The images of low probability tend to have clearer imagery features enabling easy recognition, while the images of high probability tend to be more obscure. This result indicates that the sampling schedule learned by AutoSampling may possess some hard samples mining ability.

We also draw the comparison between the sampling frequency of each training image and its loss values of different training epochs on CIFAR-100. As shown in Fig. 4, across different learning stages the correlation between loss values and sampling frequencies of training data is not strong. The high chance of being sampled by AutoSampling does not necessarily lead to high loss values, which demonstrates that AutoSampling is not merely over-sampling difficult samples, and therefore has more potential beyond simple visually hard example mining. The resulting sampling schedule learned by AutoSampling might be significantly different from the one guided by loss.

In addition, we notice the images of low probability also contain low quality images. For instance, in Fig.3 the leftmost image of CAMAL class contains only legs. This shows that AutoSampling may potentially rule out problematic training data for better training.

5.8 Transfer ability of learned sampling distributions

We hypothesis that the learned sampling schedule may be independent of the model's bias, rather it is determined by the intrinsic property of the data. To valide this hypothesis, we examine the transfer ability of sampling distributions learned by AutoSampling to other network structures. Specifically, we study transfer ability between CNN-based and

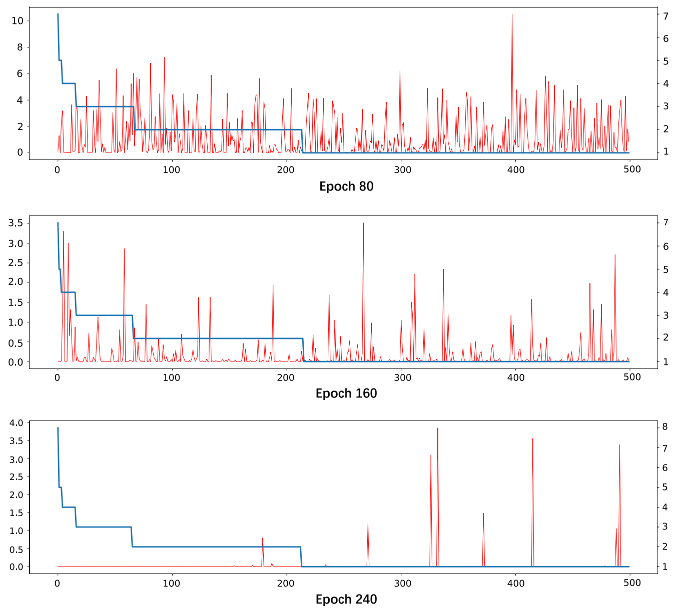


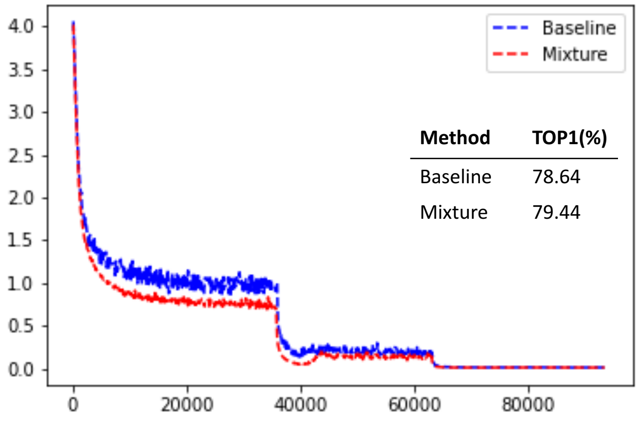
Fig. 4. The comparison between the sampling frequency of each training image and its loss values of Epoch 80, 160 and 240 from CIFAR-100 with ResNet-18. We randomly selected 500 training images, and calculate their sampling frequency and loss values. The x-axis is the indexes of 500 training images, while the left y-axis denotes loss values and the right y-axis denotes the sampling frequency. The blue line represents the sampling frequencies and the red lines represents the loss values of all 500 images. As we can see from the figure, the two lines are not obviously correlated.

Transformer-based architectures. Indeed, Transformer-based architectures have shown outstanding performance on vision tasks while having less inductive biases than convolutional neural networks. The Transformer architecture is essentially composed of two components: 1) A self attention module which essentially aggregates feature map points, 2) A feed-forward network which essentially computes the final representation from the contextual mapping. Table 3 shows that ViT-S benefits from AutoSampling.

First, we run training on ResNet-50 [43] using STATIC sampling schedule generated by three distributions learned by AutoSampling on 3 different models. As shown in Table 10, using sampling schedules learned by AutoSampling from other models, we demonstrate similar improvements over the UNIFORM baseline.

Second, we compare transfer ability across networks with different inductive bias. We use the STATIC sampling schedule generated by ViT-S and transfer it to a ResNet50. Results are shown in Table 10, where we see that ResNet-50 benefits from the sampling scheduled learned by AutoSampling on ViT-S. Conversely, we use the STATIC sampling schedule generated by ResNet50 and transfer it to a ViT-S. Transferring the sampling schedule from ResNet-50 to ViT-S does not yield significant improvements. This may be due to the fact that ViT-S is fine-tuned using a well tuned pre-trained backbone.

These results show generalization across model structures, combined with the above observations on images of different sampling probability, indicates that there may exist a common optimal sampling schedule determined by the intrinsic property of the data rather than the model being optimized. Our AutoSampling is an effort to gradually converge to such an optimal schedule.



Method	TOP1(%)
Baseline	78.64
Mixture	79.44

Fig. 5. Convergence curves of ResNet-18 with (Red) and without (Blue) using AutoSampling on CIFAR-100.

5.9 Additional findings.

AutoSampling provides faster convergence as seen on Fig. 5. To examine our method more closely, we removed the learning rate decay in the last training stage. We observed that while the last training stage with the lowest learning rate is key for the baseline it is not required when using AutoSampling. This implies that data sampling can play an important role in optimization.

5.10 Discussions

The experimental results and observations from Section 5.6 and 5.7 shed light on the possible existence of an optimal sampling schedule, which relies only on the intrinsic property of the data and the learning stage of the model, regardless of the specific model structure or any human prior knowledge. The AutoSampling method is able to provide relatively enough rewards in the searching process compared to other related works, leading to sufficient convergence towards the desired sampling schedule. Once obtained, the desired sampling schedule may also be generalized over other model structures for robust training, as shown in Table 10. Although AutoSampling requires relatively large amount of computing resources to find a robust sampler, we want to point out that the efficiency of our method can be improved through better training techniques. Moreover, the possibility of an optimal sampling schedule relying solely on the data themselves may indicate more efficient sampling policy search algorithms, if one can quickly and effectively determine data value based on its property.

6 Conclusion

In this paper, we introduce a new search based AutoSampling scheme to overcome the issue of insufficient rewards for optimizing high-dimensional sampling hyper-parameter by utilizing a shorter period of reward collection. In particular, we leverage the population-based training framework [18]. We use a shortened exploitation interval to search in the local data space and provide sufficient rewards. For the exploration step, we estimate sampling distribution from the searched sampling schedule and perturb it to search in the distribution space. We test our method on CIFAR-10/100 and ImageNet datasets [14], [30] with different networks show that it consistently outperforms the baseline methods across different benchmarks.

Acknowledgments

The authors would like to thank Matthieu Lin for reproducing the AutoSampling algorithm and providing the codebase.

References

[1] A. Estabrooks, T. Jo, and N. Japkowicz, “A multiple resampling method for learning from imbalanced data sets,” *Computational Intelligence*, 2004.
[2] G. M. Weiss, K. McCarthy, and B. Zabar, “Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs?” *Dmin*, 2007.
[3] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Int. Conf. Mach. Learn.*, 2009.
[4] T. B. Johnson and C. Guestrin, “Training deep models faster with robust, approximate importance sampling,” in *Adv. Neural Inform. Process. Syst.*, 2018.
[5] A. Katharopoulos and F. Fleuret, “Not all samples are created equal: Deep learning with importance sampling,” in *Int. Conf. Mach. Learn.*, 2018.
[6] A. Shrivastava, A. Gupta, and R. Girshick, “Training region-based object detectors with online hard example mining,” in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.
[7] A. Jesson, N. Guizard, S. H. Ghalehjegh, D. Goblot, F. Soudan, and N. Chapados, “Cased: Curriculum adaptive sampling for extreme data imbalance,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2017.
[8] V. Birodkar, H. Mobahi, and S. Bengio, “Semantic redundancies in image-classification datasets: The 10% you don’t need,” *arXiv preprint arXiv:1901.11409*, 2019.
[9] Z. Li, Y. Wu, K. Chen, Y. Wu, S. Zhou, J. Liu, and J. Yan, “Learning to auto weight: Entirely data-driven and highly efficient weighting framework,” in *Assoc. Adv. Artif. Intell.*, 2020.
[10] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei, “Mentornet: Regularizing very deep neural networks on corrupted labels,” *arXiv preprint arXiv:1712.05055*, 2017.
[11] Y. Fan, F. Tian, T. Qin, J. Bian, and T.-Y. Liu, “Learning what data to learn,” *arXiv preprint arXiv:1702.08635*, 2017.
[12] I. Loshchilov and F. Hutter, “Online batch selection for faster training of neural networks,” *arXiv preprint arXiv:1511.06343*, 2015.
[13] Y. Wang, W. Gan, J. Yang, W. Wu, and J. Yan, “Dynamic curriculum learning for imbalanced data classification,” in *Int. Conf. Comput. Vis.*, 2019.
[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2009.
[15] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, “Autoaugment: Learning augmentation policies from data,” *arXiv preprint arXiv:1805.09501*, 2018.

[16] X. Zhang, Q. Wang, J. Zhang, and Z. Zhong, "Adversarial autoaugment," in *Int. Conf. Learn. Represent.*, 2020.
[17] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams, "Scalable bayesian optimization using deep neural networks," in *Int. Conf. Mach. Learn.*, 2015.
[18] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, "Population based training of neural networks," *arXiv preprint arXiv:1711.09846*, 2017.
[19] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of Machine Learning Research*, 2012.
[20] D. Ho, E. Liang, X. Chen, I. Stoica, and P. Abbeel, "Population based augmentation: Efficient learning of augmentation policy schedules," in *Int. Conf. Mach. Learn.*, 2019.
[21] M. Sun, H. Dou, B. Li, J. Yan, W. Ouyang, and L. Cui, "Autosampling: Search for effective data sampling schedules," in *Int. Conf. Mach. Learn.*, 2021.
[22] C. Drummond, R. C. Holte *et al.*, "C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling," in *Workshop on Learning from Imbalanced Datasets II*, 2003.
[23] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Int. Conf. Comput. Vis.*, 2017.
[24] J. Byrd and Z. Lipton, "What is the effect of importance weighting in deep learning?" in *Int. Conf. Mach. Learn.*, 2019.
[25] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, "Class-balanced loss based on effective number of samples," in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.
[26] X. Wu, E. Dyer, and B. Neyshabur, "When do curricula work?" *arXiv preprint arXiv:2012.03107*, 2020.
[27] M. Ren, W. Zeng, B. Yang, and R. Urtasun, "Learning to reweight examples for robust deep learning," in *Int. Conf. Mach. Learn.*, 2018.
[28] M. MacKay, P. Vicol, J. Lorraine, D. Duvenaud, and R. Grosse, "Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions," *arXiv preprint arXiv:1903.03088*, 2019.
[29] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," in *International Conference on Artificial Intelligence and Statistics*, 2020.
[30] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Master's thesis, University of Tront*, 2009.
[31] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," in *IEEE Conf. Comput. Vis. Pattern Recog. Worksh.*, 2004.
[32] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3D object representations for fine-grained categorization," in *Int. Conf. Comput. Vis. Worksh.*, 2013.
[33] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft," *arXiv preprint arXiv:1306.5151*, 2013.
[34] I. J. Goodfellow, D. Erhan, P. L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee *et al.*, "Challenges in representation learning: A report on three machine learning contests," in *International Conference on Neural Information Processing*, 2013.
[35] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.
[36] J. Xiao, K. A. Ehinger, J. Hays, A. Torralba, and A. Oliva, "Sun database: Exploring a large collection of scene categories," *Int. J. Comput. Vis.*, 2016.
[37] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.
[38] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.
[39] X. Chen, C.-J. Hsieh, and B. Gong, "When vision transformers outperform resnets without pre-training or strong data augmentations," *arXiv preprint arXiv:2106.01548*, 2021.
[40] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.
[41] M. Zhang, J. Lucas, J. Ba, and G. E. Hinton, "Lookahead optimizer: k steps forward, 1 step back," in *Adv. Neural Inform. Process. Syst.*, 2019.
[42] X. Jin, B. Peng, Y. Wu, Y. Liu, J. Liu, D. Liang, J. Yan, and X. Hu, "Knowledge distillation via route constrained optimization," in *Int. Conf. Comput. Vis.*, 2019.
[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.
[44] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017.
[45] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.
[46] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Eur. Conf. Comput. Vis.*, 2018.