SampleAhead: Online Classifier-Sampler Communication for *Learning from Synthesized Data*

Qi Chen, Weichao Qiu, Yi Zhang
{qchen42,wqiu7,yzh}@jhu.edu
Lingxi Xie(✉), Alan L. Yuille
198808xc@gmail.com,alan.yuille@jhu.edu

Department of Computer Science,
The Johns Hopkins Univerisity

* This work is supported by IARPA via DOI/IBC contract #D17PC00345 and ONR grant N00014-15-1-2356.

Abstract

State-of-the-art techniques of computer vision are mostly data-driven, but collecting and manually labeling a large scale dataset is both difficult and expensive. A promising alternative is to use synthesized training data, so that the dataset size can be significantly enlarged with little human labor. But, this raises an important problem: given an **infinite** data space, how to effectively sample a **finite** subset to train a visual recognition model?

This paper presents an approach for learning from synthesized data effectively. The motivation is straightforward – increasing the probability of seeing difficult training data. We introduce a module named **SampleAhead** to formulate the learning process into an online communication between a *classifier* and a *sampler*, and update them iteratively. In each round, we adjust the sampling distribution according to the classification results, and train the classifier using the data sampled from the updated distribution. Experiments are performed by introducing synthesized images rendered from ShapeNet models to assist PASCAL3D+ classification. Our approach enjoys higher classification accuracy, especially in the scenario of a limited number of training samples. This demonstrates its efficiency in exploring the infinite data space.

1 Introduction

Recent progress in computer vision has been boosted by deep neural networks trained with a large amount of labeled data. Researchers made every effort to increase the volume [8, 33] and representativeness [9] of these datasets, however, the collection and annotation remain labor-intensive and error-prone. A smart idea to address this problem is to generate synthesized data (*e.g.*, from a virtual world [3, 23]) with a minimal amount of human labor.

But, because the synthesized environment allows us to sample an **infinite** amount of data, an important yet unstudied problem is raised: given a constrained time, how to effectively sample a **finite** subset so as to maximize the performance of a vision system? We address this problem with object pose estimation, a fundamental task in 3D computer vision. Note that for some specific tasks such as object pose estimation, integrating synthesized data helps a lot in improving recognition accuracy, but previous approaches often sampled data uniformly from the synthesized space [23], leading to a redundant set of *easy* training cases, while the *hard* cases cannot get trained sufficiently.

Inspired by previous work [25] which adjusted data weights according to their difficulties in an online manner, we suggest a learning system which is composed of two components, with a *classifier* (parameterized by a set of network weights θ) dealing with the recognition task, and a *sampler* (parameterized by a class distribution $P(\cdot)$ over viewpoint parameters, *e.g.*, azimuth and elevation angles) sampling training data from the infinite data space. The major algorithm for optimization is similar to AdaBoost [11], *i.e.*, increasing the weight of difficult samples in training the classifier.

The training process involves updating θ and $P(\cdot)$ in an iterative manner. The unit that controls the classifier-sampler communication is named **SampleAhead**. In each iteration, the distribution $P(\cdot)$ is determined by the testing results in a standalone validation set, and then used to sample a new batch of data for training the classifier (updating the parameter θ). To improve computational efficiency, we partition the entire space into a finite number of *buckets*. In each training epoch, the classifier is first applied on a validation set to estimate the difficulty of each bucket, and the sampler follows to construct a new training subset. This is a two-stage sampling process. Every time, a bucket is first sampled from the distribution $P(\cdot)$, and then a datum is sampled from the bucket following a uniform distribution. This iteration continues until the maximal number of rounds is reached.

We conduct experiments in a challenging task known as object pose estimation, which aims at predicting the viewpoint from which we capture a 2D image of an object. We use PASCAL3D+ [34] as the target (testing) dataset, and render a large number of synthesized images from ShapeNet [4]. In comparison to the baseline approach [28] which always sampled the data space from a fixed distribution, our method produces higher recognition accuracy especially in more challenging scenarios, in agreement with our motivation. In particular, when the number of extra training cases is limited, the advantage of our approach becomes even more significant.

The remainder of this paper is organized as follows. Section 2 briefly reviews related work, and Section 3 illustrates our overall framework as well as the joint optimization approach. After experiments are shown in Section 4, we conclude this work in Section 5.

2 Related Work

Training machine learning systems for computer vision tasks, especially deep neural networks, often requires sufficient data to prevent over-fitting. The availability of large-scale datasets facilitates the ability of training very deep neural networks [17]. However, researchers often required a considerable amount of labor to collect and annotated a large-scale dataset [8, 33], or a smaller one with reasonable variability [9, 34].

On the other hand, the rapid development of computer graphics allows researchers to construct an unreal environment [22, 35], and sample a large number of annotated synthesized data with little human labor [4, 16]. Another possibility is to apply generative deep learning models to simulate the distribution of real data [13]. It has been verified that synthesized [6, 23, 28, 31] or generated [26] data are helpful in training better models. However, in either case, we are provided with an infinite space of training data, and facing the issue of making use of these synthesized data in a constrained time, *i.e.*, the number of sampled data is finite. A related area to this problem is named Active Vision [1, 2], in which one is allowed to manipulate the viewpoint of the camera(s) in order to explore and learn richer visual knowledge from the environment. Recently, this idea was also applied to train robots in the task of visual question answering [7, 14].

There exist several ways of sampling training data from a given distribution. A straightforward solution is bootstrapping, which sampled training data with replacement. Researchers soon developed other algorithms to increase the probability of sampling a hard example, such as AdaBoost [11] and a series of negative example mining methods [24, 29] to assist training in SVM [10] and CNN [15]. At a finer level, it is also possible to adjust the weights of different elements, so that the loss function would lean towards penalizing the errors in hard examples [19, 25, 27, 32]. All these approaches were verified to outperform uniform sampling, especially when the easy examples occupy a considerable fraction of the data space.

In this paper, we focus on a more efficient sampling strategy. Different from previous work, we consider an infinite (continuous) data space. Instead of sampling from each instance (*e.g.*, an image [27] or a regional feature [25]), we partition the entire data space into a finite number of buckets and perform two-stage sampling, detailed in Section 3.3.

3 Approach

3.1 Background and Motivation

The goal of this work is to train an effective vision model from an infinite synthesized dataset. Throughout this paper, we assume the target model to be a classifier, denoted by $\mathbb{C}$: $\mathbf{Y} = \mathbf{f}(\mathbf{X};\theta)$, where $\mathbf{X}$ and $\mathbf{Y}$ are the input and output, *e.g.*, an image matrix and a one-hot vector, and θ are the parameters in the model $\mathbf{f}(\cdot)$, *e.g.*, the weights of a neural network.

Training data are sampled from an image space $\mathcal{X}$. The sampling process is a function $\mathbf{X} = \mathbf{g}(\mathbf{U})$, where $\mathbf{U}$ are the parameters (*e.g.*, object position, viewpoint, lighting, *etc.*) required by the generator $\mathbf{g}(\cdot)$. Note that $\mathbf{U}$ is sampled from the parameter space $\mathcal{U}$, which is continuous and thus infinite. **The core of this paper is to sample a number of U's at each training iteration.** Following a large corpus of previous work, we assume that each $\mathbf{U}$ is sampled independently and identically from a distribution $P(\cdot)$ defined in the parameter space $\mathcal{U}$. We denote the process of generating a training data by $\mathbf{X} = \mathbf{g}(\mathbf{U}); \mathbf{U} \sim P$.

A naive example is to set $P(\cdot)$ to be a uniform distribution over $\mathcal{U}$, *i.e.*, $P(\mathbf{U}) \equiv \text{const}$ for all $\mathbf{U} \in \mathcal{U}$. This is equivalent to generating a sufficient large synthesized dataset at the beginning, and traverse each item orderly. However, in most scenarios, the classifier are dealing with relatively easy training cases, *e.g.*, those cases that are already been correctly classified, so that the weights θ cannot get trained efficiently.

3.2 The SampleAhead Module

To deal with the above issue, we introduce a module named **SampleAhead**. This module updates the data distribution $P(\cdot)$ *before* each sampling stage (in practice, before each training epoch), increasing the probability that hard examples are sampled and fed into the classifier.

Ideally, at the t-th iteration, for each sample $\mathbf{U}$, we hope that $P^{(t)}(\mathbf{U})$ tends to have peaks at the *hard* cases. We start with defining the *difficulty* of $\mathbf{U}$, denoted by $d^{(t-1)}(\mathbf{U})$, as the probability that $\mathbf{X} = \mathbf{g}(\mathbf{U})$ is not correctly classified by the classifier after the $t-1$-st iteration. However, directly computing $d^{(t-1)}(\mathbf{U})$ for every $\mathbf{U}$ could be sensitive to noise. We make use of kernel estimation, which randomly distributes a set of M probes $\mathcal{V} = \{\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_M\}$ over the entire space $\mathcal{U}$, and estimate the difficulty of $\mathbf{U}$ by:

$$d^{(t-1)}(\mathbf{U}) = \frac{\sum_{m=1}^{M} d^{(t-1)}(\mathbf{V}_m) \cdot \omega(\mathbf{U}, \mathbf{V}_m)}{\sum_{m=1}^{M} \omega(\mathbf{U}, \mathbf{V}_m)}, \tag{1}$$

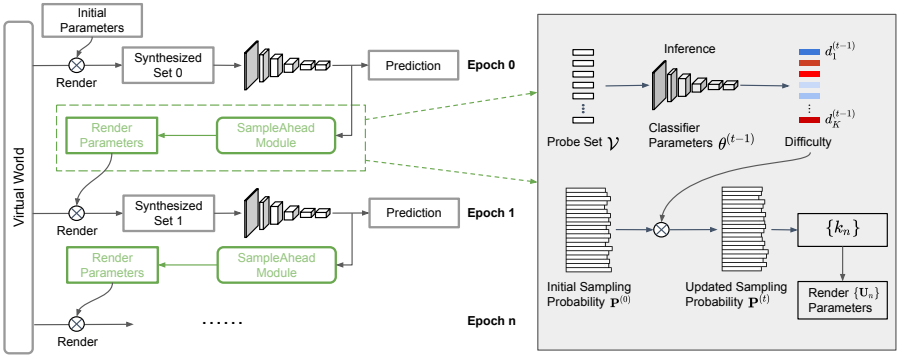

Figure 1: The overall framework of our approach. Without the SampleAhead module, our approach degenerates to that fixing a synthesized dataset at the very beginning and traversing all training samples orderly. Each SampleAhead module conducts an interactive process between the classifier and the sampler, in which the classification results in a standalone probe set are used to estimate the new training data distribution, from which the sampler generates the new set for the next training epoch.

where

$$d^{(t-1)}(\mathbf{V}_m) = 1 - \Pr\left[\mathbf{f}\left(\mathbf{g}(\mathbf{V}_m); \theta^{(t-1)}\right) \text{ is correct}\right], \tag{2}$$

and $\omega(\mathbf{U}, \mathbf{V}_m)$ is the weight added to $\mathbf{U}$ by the probe $\mathbf{V}_m$, *e.g.*, $\omega(\mathbf{U}, \mathbf{V}_m)$ is inversely proportional to the ℓ_2-distance between $\mathbf{U}$ and $\mathbf{V}_m$. The probe set $\mathcal{V}$ is often large in order to guarantee the coverage over the space $\mathcal{U}$.

The next step is to define the probability distribution function $P^{(t)}(\mathbf{U})$ for each $\mathbf{U} \in \mathcal{U}$. Inspired by AdaBoost [□], we take the classification results in the previous iteration into consideration. Mathematically,

$$P^{(t)}(\mathbf{U}) \propto \alpha \cdot P^{(0)}(\mathbf{U}) + (1-\alpha) \cdot P^{(0)}(\mathbf{U}) \cdot e^{\beta \cdot d^{(t-1)}(\mathbf{U})}, \tag{3}$$

where α and β are hyper-parameters. We use $P^{(0)}(\cdot)$ rather than $P^{(t-1)}(\cdot)$ to avoid the distribution from being modified too much. This strategy improves training stability.

3.3 Approximation

Note that accurately sampling from $P^{(t)}$ in Eqn (3) requires computing the function value at each $\mathbf{U}$, which is computationally intractable given a large M. Here we provide an approximation for efficient online sampling. The basic idea is to partition the entire space $\mathcal{U}$ into a finite number (K) of *buckets*, *i.e.*, $\mathcal{U} = \bigcup_{k=1}^{K} \mathcal{B}_k$. Each bucket is a continuous subset of $\mathcal{U}$, and any two different buckets do not intersect with each other. Thus, we simplify Eqn (1) by only considering the probes in the same bucket, namely,

$$w(\mathbf{U}, \mathbf{V}_m) = \mathbb{I}[\exists k; \mathbf{U} \in \mathcal{B}_k \wedge \mathbf{V}_m \in \mathcal{B}_k], \tag{4}$$

where $\mathbb{I}[\cdot]$ is the indicator function. Note that for any $\mathcal{B}_k$, every element $\mathbf{U} \in \mathcal{B}_k$ has the same distance to each probe, thus the same difficulty $d_k^{(t-1)}$ (omitting $\mathbf{U}$):

$$d_k^{(t-1)} = \frac{\sum_{\mathbf{V}_m \in \mathcal{B}_k} d^{(t-1)}(\mathbf{V}_m)}{|\mathcal{V} \cup \mathcal{B}_k|}. \tag{5}$$

This actually leads to **a two-stage sampling process**, in which a bucket-level probability is computed for each bucket:

$$P_k^{(t)} = \int_{\mathbf{U}\in\mathcal{B}_k} P^{(t)}(\mathbf{U})\,\mathrm{d}\mathbf{U}. \tag{6}$$

Every time we hope to generate a $\mathbf{U}\in\mathcal{U}$, we first determine the bucket index k from a finite set $\{1,2,\ldots,K\}$, and then sample a $\mathbf{U}$ from $\mathcal{B}_k$ following a uniform distribution.

In practice, we update $P_k^{(t)}$ after each training epoch. This is not done after each mini-batch because of its large computational costs (requiring a complete testing in the probe set which is often large). At the beginning, $P_k^{(0)}$ is simply defined as the probability that a uniform sampling in $\mathcal{U}$ falls into $\mathcal{B}_k$. In updating $P_k^{(t)}$ with Eqn (3), note that both $P^{(0)}(\mathbf{U})$ and $d^{(t-1)}(\mathbf{U})$ are constants within $\mathcal{B}_k$, thus Eqn (3) is simplified as:

$$P_k^{(t)} = \alpha\cdot P_k^{(0)} + (1-\alpha)\cdot P_k^{(0)}\cdot e^{\beta\cdot d_k^{(t-1)}}. \tag{7}$$

The definition of buckets differs from case to case, and will be discussed in experiments.

Of course, there are some technical details that can be discussed, such as sharing/reusing data in the training set and the probe set, which we will investigate in the future.

4 Experiments

4.1 MNIST: Digit Classification

- **Dataset and Settings**

We first evaluate our approach on a toy problem, which is handwritten digit classification on the MNIST dataset [18]. MNIST contains 60,000 training images and 10,000 testing images. The resolution of each image is 28×28. We use this relatively simple dataset to observe the behavior of our approach on a series of data augmentation as well as discover the advantage of our approach with respect to the number of training samples.

Following [6], we consider seven types of augmentation, including digit rotation, vertical/horizontal scaling, horizontal/vertical shifting, and horizontal/vertical shearing. Each digit is processed by one and exactly one augmentation. We further partition each type into a finer stage according to the transformation parameter. The rotation angle is randomly sampled from $[-15^\circ, 15^\circ]$, and it is divided into four bins $[-15^\circ,-7.5^\circ)\cup[-7.5^\circ,0^\circ)\cup[0^\circ,7.5^\circ)\cup[7.5^\circ,15^\circ]$. All other scaling/shifting/shearing parameters are divided into two bins. Thus, we obtain 16 bins for each original training image.

The bucket set $\{\mathcal{B}_k\}_{k=1}^K$ is the Cartesian product of the class set (10 elements) and the bin set (16 elements), *i.e.*, there are $10\times 16 = 160$ buckets in total. This is to say, we assume, by Eqn (4), that all samples with the same class and a similar transformation share the same difficulty, which is reasonable. We randomly sample 100 images from each bucket to compose of the probe set $\mathcal{V}$ (16,000 probes in total).

- **Results and Analysis**

We use LeNet [18] as the classifier $\mathbf{f}(\cdot;\theta)$. It contains 3 convolutional, 2 pooling and 2 fully-connected layers. The network is trained with Stochastic Gradient Descent. Each iteration contains a mini-batch of size 76 (60 real data and 16 synthesized data). The initial learning rate is 5×10^{-3}, decayed with the `inv` policy, and the weight decay is fixed to be 5×10^{-4}. The original training process lasts for 10,000 iterations, but we allow a larger

# of Iterations	10,000	20,000	30,000	40,000	50,000
Uniform Samp.	0.890 ± 0.021	0.835 ± 0.024	0.816 ± 0.021	0.796 ± 0.018	0.793 ± 0.035
Our Approach	0.819 ± 0.025	0.787 ± 0.019	0.756 ± 0.022	0.758 ± 0.026	0.757 ± 0.014
p-value	6.407×10^{-5}	2.434×10^{-3}	4.454×10^{-4}	8.097×10^{-3}	3.054×10^{-2}

Table 1: Classification error rates (%) on the MNIST dataset with respect to the number of sampled images. The average and standard deviation numbers come from 10 individual runs. The p-values are obtained from standard t-tests over these 10 pairs.

number (20,000, 30,000, 40,000 and 50,000) of iterations so that more augmented data are seen. Without synthesized data, the error rate hardly goes down after 10,000 iterations.

Results are summarized in Table 1. We can observe that our approach works consistently better than the baseline approach (performing uniform sampling in the data space). In particular, when the model is constrained to see a limited amount of training data, the advantage of our approach becomes even more significant, *e.g.*, with 10,000 iterations, the absolute and relative error rate drops brought by our approach are 0.071% and 7.98%, respectively, with a p-value of 6.407×10^{-5}, demonstrating strong statistical significance. This implies that our approach explores the data space more efficiently by aggressively looking for those challenging training samples. However, as the number of training data increases, the advantage becomes smaller, *e.g.*, with 50,000 iterations, the absolute and relative error rate drops are 0.034% and 4.29%, respectively, with a p-value of 3.054×10^{-2} which still suggests statistical significance. This is MNIST is relatively simple: given a sufficient amount of training data, random sampling can gradually achieve comparable performance to our approach.

4.2 ShapeNet: Object Pose Estimation

- **Dataset and Settings**

We move to a natural image dataset named PASCAL3D+ [34], a challenging corpus for 3D object detection and pose estimation. The 12 rigid object classes (with more than 3,000 images per class) in the PASCALVOC dataset [9] were augmented with 3D annotations, exhibiting more variability than other 3D datasets.

Due to the limited amount of data, we follow a recent baseline named RenderForCNN [28] which generated synthesized data to assist network training. To construct an augmented training set, a joint distribution of viewpoint angles and camera distances was first estimated from the PASCAL3D+ *real* training set, and 2.4 million synthesized images were rendered from 3D models of ShapeNet [4] following the same distribution. This is to say, the data distribution is fixed throughout the entire training process. Differently, we add the SampleAhead module (Eqn (7)) to enable updating data distribution according to validation.

Note that α in Eqn (7) controls the fraction of newly generated data. Setting $\alpha = 1.0$ causes our algorithm degenerate to the baseline, *i.e.*, freezing the distribution $\mathbf{P}^{(t)} \equiv \mathbf{P}^{(0)}$ throughout the entire training process. In practice, we set $\alpha = 0.9$ to take advantage of new data meanwhile preventing the training process from being slowed down by the time-consuming data generation (image rendering) process.

Based on these settings, we perform two challenging tasks, known as *object-detection-and-pose-estimation* [28] and *viewpoint prediction* [30].

- **Object Detection and Pose Estimation**

In the first task, the system is asked to detect the object and estimate its azimuth view

Approach	L	*aero.*	*bicy.*	*boat*	*bus*	*car*	*chair*	*table*	*moto.*	*sofa*	*train*	*tv*	**mean**
[28]	4	54.0	50.5	15.1	57.1	41.8	15.7	18.6	50.8	28.4	46.1	58.2	39.7
Baseline	4	62.2	59.0	17.6	61.6	**48.2**	17.2	**20.5**	**60.0**	**34.9**	**51.6**	**60.5**	44.8
Ours	4	**63.2**	**59.8**	**18.8**	**62.7**	47.1	**19.7**	18.6	59.7	34.2	51.2	59.7	**45.0**
[28]	8	44.5	41.1	10.1	48.0	36.6	13.7	15.1	39.9	26.8	39.1	46.5	32.9
Baseline	8	55.8	52.2	14.7	48.2	42.9	15.5	**16.6**	51.0	29.3	**48.0**	45.9	38.2
Ours	8	**60.0**	**52.6**	**15.2**	**53.6**	**44.2**	**18.6**	15.3	**53.1**	**31.2**	47.4	**49.9**	**40.1**
[28]	16	27.5	25.8	6.5	45.8	29.7	8.5	12.0	31.4	17.7	29.7	31.4	24.2
Baseline	16	39.8	34.3	**9.4**	51.8	35.5	11.9	**17.6**	36.3	20.3	35.2	29.9	29.3
Ours	16	**43.6**	**40.0**	8.7	**57.2**	**38.6**	**14.9**	15.1	**37.5**	**23.7**	**35.6**	**36.0**	**31.9**
[28]	24	21.5	22.0	4.1	38.6	25.5	7.4	11.0	24.4	15.0	28.0	19.8	19.8
Baseline	24	28.4	23.4	7.7	39.5	32.1	10.6	**12.6**	28.1	19.7	38.5	17.9	23.5
Ours	24	**37.9**	**30.3**	**8.7**	**47.4**	**33.9**	**13.5**	10.9	**28.7**	**22.8**	**39.0**	**26.3**	**27.2**

Table 2: Accuracy (%) of object detection and pose estimation. L is the number of azimuth bins. A testing result is accepted if both the class and pose are predicted correctly. We use a later version released by the same authors of [28] as our "Baseline", which performs considerably better than the original version. The *bottle* class is not included because its azimuth angle is almost unrecognizable.

angle simultaneously (the elevation view angle is not considered). Following [34], the output is considered correct if it is accepted by both object detection and pose estimation. The correctness object detection is measured by the IOU between the predicted and ground-truth bounding boxes. For view angle prediction, we partition the entire 360° azimuth range into 4, 8, 12 and 24 bins, and compute the accuracy that the predicted angle falls into the same bin as the ground-truth angle. 1 out of of the 12 classes (*bottle*) is not evaluated in this task, as the azimuth angle of such objects is unrecognizable.

The synthesized training set contain 3D objects captured from an azimuth angle of $[0°, 360°)$ and an elevation angle of $[-90°, 90°)$. We partition the viewpoint hemisphere into 18×12 bins of an equal size. Adding the 11 classes, we have $11 \times 18 \times 12 = 2376$ buckets in total. The validation subset from the PASCAL3D+ is used as the probe set $\mathcal{V}$.

Following the baseline [28], we extract region proposals from RCNN [12], and use these images to train an AlexNet [17] for joint object and viewpoint classification ($11 \times L$ classes, $L = 4, 8, 12, 24$). All technical details (learning rate, weight decay, *etc.*) remain the same as the baseline. We train the network for 60,000 iterations, while the baseline needs 120,000 iterations to traverse all synthesized images. We do not update data distribution (performing uniform sampling) in the first 8,000 iterations so as to provide a stable initialization.

Results are summarized in Table 2. In terms of average accuracy (the last column), our approach outperforms the baseline in every single task. Note that we only use half the number of iterations compared to the baseline, which demonstrates a favorable efficiency in exploring the infinite data space. Note that our baseline used on old-styled detector (RCNN) and classifier (AlexNet) which limited its accuracy, yet recent work [20, 21] reported higher accuracy than our work with stronger backbones, *e.g.*, [20] used Fast-RCNN for detection and VGGNet for classification. We chose to report on the same network configuration in order to make fair comparison to our baseline [28]. Yet our approach is easily generalized to a wide range of network architectures.

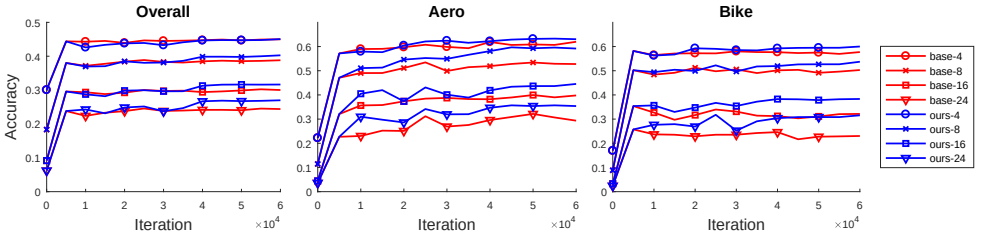

Figure 2: Compared to the baseline, our approach works better in the situation of limited data sampling. The first 8,000 iterations are shared by both approaches. Each number in the legend indicates the number of bins (L) in the classification task.

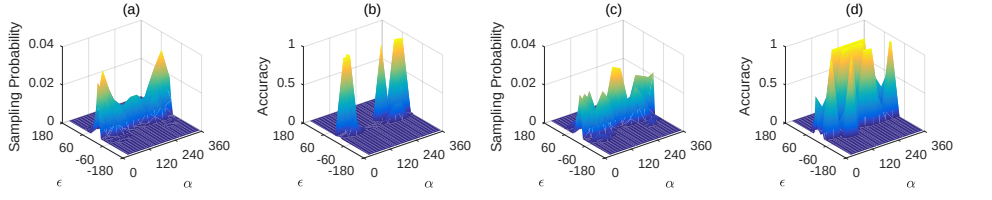

Figure 3: Comparison of sampling probability and accuracy between baseline (left two) and our approach (right two). The target class is *bike*, and α and ε on the axes denote azimuth and elevation angles, respectively. (a) The baseline simply follows the distribution in the synthesized set, without "realizing" that (b) the center part ($\alpha \approx 180°$) is more difficult. (c) On the other hand, our approach samples a larger amount of data at this area, leading to (d) a significant accuracy gain (*e.g.*, see Table 2, 30.8% vs. 23.4% in 24-bin classification).

An interesting property of our approach is the increase in accuracy gain when the number of bins L goes up. As shown in Figure 2, this happens in both the overall accuracy and individual classes *e.g.*, *bike*. This is a side benefit brought by our approach, which mines more difficult examples to improve the performance in these challenging tasks.

We diagnose our approach with additional experiments. Our approach mainly benefits from two abilities, *i.e.*, updating sampling distribution during the training process and generating new data based on the updated distribution. Switching off the former ability turns it back to the baseline, with 0.1%, 0.9%, 1.7% and 1.9% accuracy drops in $L = 4, 8, 12, 24$, respectively. The benefit brought by our approach becomes more significant as the number of bins goes up (*i.e.*, the task becomes more challenging). This is qualitatively verified in Figure 3, in which our approach increases the sampling probability of the difficult buckets, thus improving the overall accuracy. On the other hand, we also disable the latter ability by only allowing our approach to sample from the original 240 million synthesized images. This causes 0.1%, 1.1%, 1.2% and 1.7% drops, respectively, because the synthesized dataset is fixed and the difficult class, when requiring more samples, may come into duplicated training data. This ablation study shows that both generating and sampling strategies are useful yet complementary to our approach.

As a final note, our approach does not work well on the class *table*, which contributes the largest deficit compared to the baseline. This class has a significant difference from others,

Approach	*aero.*	*bicy.*	*boat*	*bott.*	*bus*	*car*	*chair*	*table*	*moto.*	*sofa*	*train*	*tv*	**mean**
$\text{Acc}_{\pi/6}$ (Baseline)	0.80	**0.84**	**0.62**	**0.96**	**0.95**	0.85	0.75	**0.86**	**0.88**	**0.87**	0.82	0.90	**0.84**
$\text{Acc}_{\pi/6}$ (Ours)	**0.84**	**0.84**	0.58	**0.96**	0.92	**0.88**	**0.91**	0.57	**0.88**	**0.87**	**0.85**	**0.93**	**0.84**
MedErr (Baseline)	10.2	12.2	18.5	6.5	4.5	6.4	12.4	**8.6**	13.0	11.0	5.7	13.1	10.2
MedErr (Ours)	**8.7**	**11.5**	**18.4**	**6.4**	**2.4**	**4.5**	**7.3**	12.5	**10.5**	**8.3**	**4.4**	**9.0**	**8.6**

Table 3: Comparison on viewpoint estimation with ground-truth bounding box provided. Here, $\text{Acc}_{\pi/6}$ measures accuracy (the higher the better) and MedErr measures error (in degrees, the lower the better). Our mean $\text{Acc}_{\pi/6}$ is mainly impacted by the low value of *table* (explained in the texts), without which we outperform the baseline by 0.86 vs. 0.84.

that rotating it by 90° merely changes its appearance, thus the 4-bin viewpoint estimation is just a random guess. In this scenario, the baseline approach memorizes the data distribution, but our approach actually discards this "cheating benefit" and thus performs "a worse guess".

- **Viewpoint Estimation**

The second task is aimed at estimating the viewpoint to the target object. Following [30], we directly use the trained model previously, and remove the factor of inaccurate object detection by directly using the ground-truth bounding box for each object. Given the ground-truth azimuth, elevation and in-plane angles and the predicted values, we compute their rotation matrices $\mathbf{R}$ and $\mathbf{R}'$ accordingly, and the included angle between them is computed by $\rho = \left\|\log(\mathbf{R}^\top \mathbf{R}')\right\|_{\text{F}} / \sqrt{2}$, where $\|\cdot\|_{\text{F}}$ is the Frobenius norm. There are two metrics in evaluation. The first one, named $\text{Acc}_{\pi/6}$, computes the fraction that $\rho \leqslant \pi/6$; and the second one, named MedErr, directly measures the median ρ value in degrees.

Results are summarized in Table 3. Our mean $\text{Acc}_{\pi/6}$ value is just slightly higher than the baseline. Note that the *table* class contributes negatively for the same reason analyzed in the previous task; but in all the remaining classes, our approach performs better. The average accuracies over the remaining 11 classes are 0.86 vs. 0.84. In addition, the median estimation error MedErr is significantly reduced (a 15.7% relative drop). All these experiments verify the effectiveness of our approach in learning from synthesized data.

5 Conclusions

This paper focuses on a new problem, which aims at effectively sampling synthesized data from an **infinitely** large parameter space. Our motivation is very simple, *i.e.*, increasing the probability of generating hard examples so that the classifier gets trained better. To this end, we insert a novel module named **SampleAhead**, which maintains a distribution over the sampling space. In each training unit, the distribution is first updated according to the current recognition results, and then used to sample synthesized training data and optimize the vision system. The concept of *buckets* is introduced to accelerate this process. Although being simple, our approach works well in a challenging vision task – joint object detection and pose estimation, especially when the recognition task is difficult (*e.g.*, the number of azimuth bins is large). Our study demonstrates the effectiveness of nonuniform sampling in an infinite set, and the advantage is more significant in the scenario of less training time (*i.e.*, fewer synthesized data are sampled).

Our algorithm has potential applications in reinforcement learning in the real world. A typical setting is to place an agent (*e.g.*, a robot) in a room, and facilitate it to learn from the surrounding world by itself. Our research matches this scenario very well, since the data space is almost infinite but training time is limited.

References

[1] R. Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988.

[2] Andrew Blake and Alan Yuille. *Active vision*. MIT press, 1993.

[3] D.J. Butler, J. Wulff, G.B. Stanley, and M.J. Black. A naturalistic open source movie for optical flow evaluation. In *European Conference on Computer Vision*, 2012.

[4] A.X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[5] W. Chen, H. Wang, Y. Li, H. Su, Z. Wang, C. Tu, D. Lischinski, D. Cohen-Or, and B. Chen. Synthesizing training images for boosting human 3d pose estimation. In *International Conference on 3D Vision*, 2016.

[6] D.C. Ciresan, U. Meier, L.M. Gambardella, and J. Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12):3207–3220, 2010.

[7] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra. Embodied question answering. *arXiv preprint arXiv:1711.11543*, 2017.

[8] J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition*, 2009.

[9] M. Everingham, L. Van Gool, C.K.I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2): 303–338, 2010.

[10] P.F. Felzenszwalb, R.B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[11] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition*, 2014.

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

[14] D. Gordon, A. Kembhavi, M. Rastegari, J. Redmon, D. Fox, and A. Farhadi. Iqa: Visual question answering in interactive environments. *arXiv preprint arXiv:1712.03316*, 2017.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision*. Springer, 2014.

[16] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C.L. Zitnick, and R. Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Computer Vision and Pattern Recognition*, 2017.

[17] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[18] Y LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[19] I. Loshchilov and F. Hutter. Online batch selection for faster training of neural networks. *arXiv preprint arXiv:1511.06343*, 2015.

[20] F. Massa, R. Marlet, and M. Aubry. Crafting a multi-task cnn for viewpoint estimation. *arXiv preprint arXiv:1609.03894*, 2016.

[21] P. Poirson, P. Ammirato, C.Y. Fu, W. Liu, J. Kosecka, and A.C. Berg. Fast single shot detection and pose estimation. In *International Conference on 3D Vision*, 2016.

[22] W. Qiu and A. Yuille. Unrealcv: Connecting computer vision to unreal engine. In *Workshops on European Conference on Computer Vision*, 2016.

[23] E. Richardson, M. Sela, and R. Kimmel. 3d face reconstruction by learning from synthetic data. In *International Conference on 3D Vision*, 2016.

[24] H.A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

[25] A. Shrivastava, A. Gupta, and R. Girshick. Training region-based object detectors with online hard example mining. In *Computer Vision and Pattern Recognition*, 2016.

[26] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. In *Computer Vision and Pattern Recognition*, 2017.

[27] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, and F. Moreno-Noguer. Fracking deep convolutional image descriptors. *arXiv preprint arXiv:1412.6537*, 2014.

[28] H. Su, C.R. Qi, Y. Li, and L.J. Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *International Conference on Computer Vision*, 2015.

[29] K.K. Sung. Learning and example selection for object and pattern detection. 1996.

[30] S. Tulsiani and J. Malik. Viewpoints and keypoints. In *Computer Vision and Pattern Recognition*, 2015.

[31] G. Varol, J. Romero, X. Martin, N. Mahmood, M.J. Black, I. Laptev, and C. Schmid. Learning from synthetic humans. In *Computer Vision and Pattern Recognition*, 2017.

[32] X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. In *International Conference on Computer Vision*, 2015.

[33] Z. Wu, S. Song, A. Khosla, X. Tang, and J. Xiao. 3d shapenets for 2.5d object recognition and next-best-view prediction. *arXiv preprint arXiv:1406.5670*, 2014.

[34] Y. Xiang, R. Mottaghi, and S. Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In *Winter Conference on Applications of Computer Vision*, 2014.

[35] Yi Zhang, Weichao Qiu, Qi Chen, Xiaolin Hu, and Alan Yuille. Unrealstereo: A synthetic dataset for analyzing stereo vision. *arXiv preprint arXiv:1612.04647*, 2016.