Data Augmentation을 사용한 Image Classifier Training의 Input Pipeline Overhead 분석

이하연°[1], 이계원[2], 전병곤[2]

조지아 공과대학교 컴퓨터과학과[1]
서울대학교 컴퓨터공학부[2]
ilee300@gatech.edu[1] {gyewonlee, bgchun}@snu.ac.kr[2]

Analysis of Input Pipeline Overhead for Training Image Classifiers with Data Augmentation

Irene Lee°[1], Gyewon Lee[2], Byung-Gon Chun[2]
College of Computing, Georgia Institute of Technology[1]
Department of Computer Science and Engineering, Seoul National University[2]

요 약

Image classification is a popular application of deep neural networks. Data augmentation is often used to train models for such task in order to reduce estimation error from limited number of data and to increase generalization of trained models. However, due to the non-deterministic nature of data augmentation, preprocessed data is not reusable, leaving the input pipeline overhead unavoidable for every iteration of training. In this work, we analyze the impact of input pipeline overhead on training throughput of image classification models in various hardware settings. The key observations of the experiment are that data augmentation degrades training speed, and that the degradation becomes worse as 1) the model being trained gets smaller, 2) the CPU-GPU ratio decreases, and 3) the number of augmentation layers increases.

1. Introduction

Image classification is one of the most popular computer vision tasks that aims at identifying and classifying an image as one of the predefined categories. Trained image classifiers can be useful in other tasks such as object detection/recognition and automated image organization as well [5]. A technique that is often used when training image classifiers is data augmentation. Data augmentation, as its name denotes, augments new data from the original input training data set. This is done in order to increase the diversity in the training set and make the model robust to potential variances in test data such as flips and rotations. There has been active research regarding how to augment data more efficiently and effectively [3, 4, 6, 7, 8]. One of the most recent augmentation method, RandAugment [4], has shown an accuracy as high as 97.2 on the ImageNet data with EfficientNet-B7 model. Although the effectiveness of data augmentation on image classification tasks has been evaluated in many research, there has been little done regarding the impact of data augmentation on the training speed. Data augmentation requires CPU-intensive operations, which are usually non-deterministic, and so its CPU overhead cannot be mitigated with simple caching and reusing. As such, input pipeline overhead exists for every epoch, and this can cause the CPU to become the bottleneck in the training. Because training speed is another important factor to consider when training a model, we found it necessary to evaluate the train speed degradation with data augmentation in various hardware settings.

2. Background

In this section, we first go over the general procedure of training an image classification model, and in particular data augmentation.

2.1 Training Pipeline for Image Classification Tasks

Training pipeline refers to the overall process of training a model. There are three main stages in training pipeline: loading the data, preprocessing the input data, and training the model. The first step, loading, refers to reading in the input dataset stored in the disk. If the size of the entire dataset is small enough to fit the memory, the whole dataset can be loaded into the memory. If not, the dataset can be loaded in batches from the disk to the memory. Next step is preprocessing the input images. This is a necessary step because the images have to be decoded and the input images can be of different sizes whereas the input tensor size of the model is usually static. There is no fixed rule for preprocessing as long as the preprocessed images fit the input tensor size. Operations done in this step can be both stochastic or deterministic. Some common procedures include random crop and flip, and normalization and standardization. Data augmentation[2.2] is often done in this step as well. Preprocessed data is then sent for training. This final step of the training pipeline updates the weights of the model's parameters via forward propagation and then backward propagation of resulting gradient of the weights.

2.2 Data Augmentation

Data augmentation is a technique used to artificially increase the number of training data in order to decrease the test error. By randomly applying distortions such as rotation, flip, and inversion to the input image, the model becomes invariant to those transformations in the test data. Some existing data augmentation methods are AutoAugment, Population Based Augmentation, RandAugment, Mixup, and CutMix[3, 6, 4, 8, 7]. AutoAugment uses a policy that consists of many sub-policies from which one is chosen to be applied to each image in a mini-batch. Each sub-policy is a list of transformations to be done for each input image. The policy provides how often and how much each transformation should be done. AutoAugment uses reinforcement learning to find a policy

that yields the highest validation accuracy. To mitigate the computational strain of the reinforcement learning in AutoAugment, Population Based Augmentation was developed. This augmentation uses evolutionary algorithm to find a schedule of increasing the magnitude of the transformation being applied. RandAugment further simplifies this task by searching only for the magnitude of transformation. It randomly selects some number of transformations depending on the number of augmentation layers (i.e. the number of transformations to be applied to each image). Then each transformation is done with the same found magnitude. Mixup and CutMix augment new images by mixing up two images from the batch in a random fashion. The former randomly interpolates between two images and also between the selected images' output. For example, if two images are mixed up with .3 interpolation, then the output would be 0.3 and 0.7 for each. Similarly, CutMix cuts and pastes one image on another with some ratio, with which the output would be also interpolated.

3. Experiment

Because data augmentation is done in a random fashion, the operations are usually done in the CPU [1, 2]. This task can be done in parallel with the training in the GPU, but for the training to be done, the input data needs to be preprocessed beforehand in the CPU. Therefore, if the CPU time becomes longer than GPU, GPU will be idle, waiting for the CPU to finish the augmentation. In addition, due to the stochasticity, the augmented images are not reusable. Therefore, caching is not effective, making it necessary to perform data augmentation for every epoch. Thus, this CPU intensive input pipeline overhead can become the bottleneck of training. As the popularity of data augmentation is increasing and with proved improvements in test accuracy with its use, we found it necessary to evaluate the implication of data augmentation in training speed. We analyze the the impact of data augmentation in training speed for different models in varying hardware settings.

3.1 Experiment Setup

The experiment is conducted with CUDA 10.2 on a single NVIDIA Titan RTX GPU. We have chosen two convolutional neural network models: ResNet50 and MobileNetV2. The former is a heavy 50-layer model and the latter is a light-weight 3-layer model. These models have been selected to evaluate how the model size affects the degree to which the data augmentation degrades the training speed. The training is done using ImageNet data, with the batch size of 64 and the data type of float32. RandAugment [4] is chosen as the augmentation strategy. Using a Docker container, we have also controlled the number of available CPU cores to two, four, six, and eight, among which eight is the default. We fix the number of GPU to one. For the evaluation, we define different categories of input pipeline as the following: *Synthetic, Basic,* and *Preprocessing + RandAugment*. For *Synthetic*, a random tensor of the size of the model's input tensor is generated, and thus no preprocessing is needed at all. *Basic* refers to the basic preprocessing steps excluding data augmentation. Random crop and flip, and normalization and standardization are included in this input pipeline. Lastly, *Preprocessing + RandAugment* denotes the full input pipeline that embodies both the basic preprocessing and data augmentation. (This

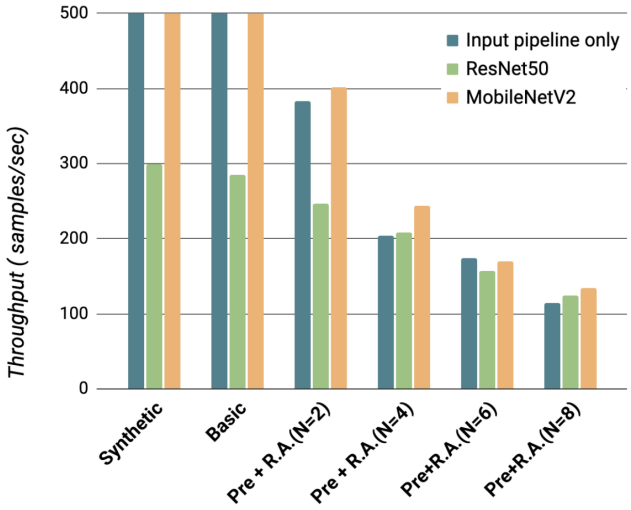

Figure 1: The end-to-end training throughput and sole input pipeline throughput. Because the throughput of Input pipeline only is much greater than the rest, the graph has been cropped at maximum throughput of 500 samples per second.

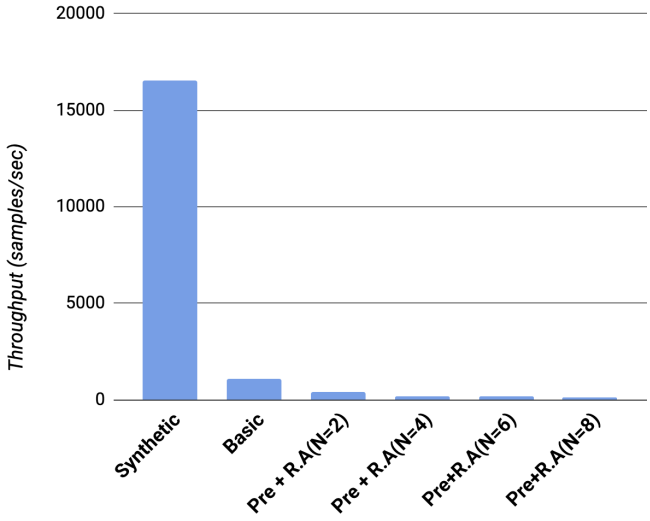

Figure 2: The end-to-end training throughput of Input pipeline alone

is abbreviated as *Pre + R.A.*) We have tested this category of input pipeline with different number of augmentation layers, N.

3.2 Results

The first setup tests the end-to-end training throughput (measured in samples per second) of the two models and the input pipeline alone in the default setting. *Synthetic* part is representative of pure GPU time, as there is no preprocessing done. Although the CPU is used to generate the random tensors, this overhead is relatively low and thus negligible. This means that the throughput for *synthetic* would be the benchmark for image preprocessing optimization, since with input pipeline throughput greater than that will make the GPU the bottleneck of the training process. On the other hand, *input pipeline only* is representative of pure CPU time, as no training is done and so the GPU is not used. *Input pipeline only* is shows the limit CPU sets on the end-to-end throughput of the

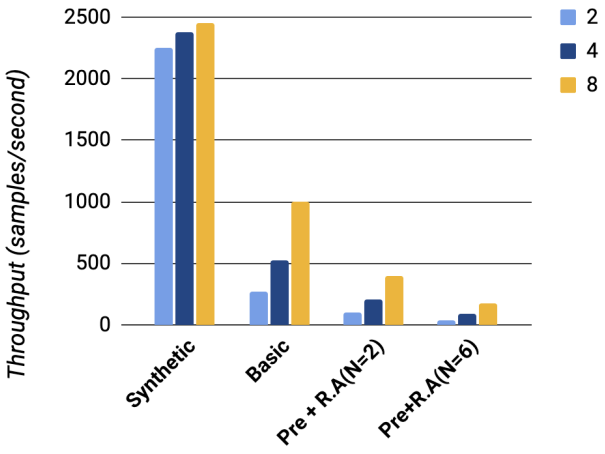

Figure 3: MobileNetV2 end-to-end training throughput with different CPU-GPU ratios

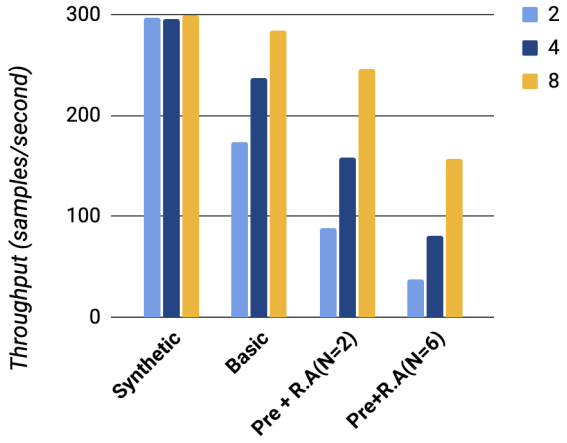

Figure 4: ResNet50 end-to-end training throughput with different CPU-GPU ratios

models. In general, when the *input pipeline only* throughput goes below the throughput of *synthetic* for any model, its end-to-end training throughput decreases. This is natural, since the GPU has to wait for the preprocessed batch to come from the CPU for each iteration. Slower CPU operation causes the GPU to wait longer, decreasing the average throughput of the overall training pipeline. Also, the image preprocessing becomes the bottleneck of the system with smaller number of augmentation layers for MobileNetV2 than ResNet50. Since the pure GPU throughput is already high enough for the smaller model, slight decrease in the CPU throughput is enough for the CPU become the bottleneck of the overall system. This observation can be generalized to the conclusion that light-weight models are more susceptible to input-pipeline overhead than heavier ones.

The next set of experiments evaluates how much data augmentation influences the end-to-end throughput in different hardware settings with varying CPU-GPU ratio. The results show that the degree to which the training throughput degrades becomes less severe as the CPU-GPU ratio increases. The main difference between the results from ResNet50 and MobileNetV2 is that the throughput degradation of MobileNetV2 is much more drastic than that of the other. This is because MobileNetV2 is a much smaller model than ResNet50, and thus more sensitive to the throughput change in the CPU.

4. Conclusion

Data augmentation is a popular technique used to train image classification models. It artificially increases the dataset and the diversity in the dataset, making the model be more robust to variances such as rotations and flips. However, due to the stochasticity of the technique, caching cannot be done and the CPU intensive preprocessing job has to be done for every epoch. In this paper, we evaluate the impact of input pipeline overhead posed by data augmentation on the training throughput of image classifiers in various hardware settings. Data augmentation decreases the overall throughput of the end-to-end training, and the degradation becomes worse as the number of augmentation layer increases. Smaller models such as MobileNetV2 are more susceptible to throughput degradation due to data augmentation than larger models like ResNet50. In addition, as the CPU-GPU ratio increases, the degree to which the overall throughput degrades decreases.

5. Acknowledgement

This work was supported by Institute of Information communications Technology Planning Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2015-0-00221, (SW 스타랩) 다양한 분석을 고속 수행하는 단일화된 빅데이터 스택 개발)

References

[1] Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. `https://github.com/pytorch/pytorch`.

[2] Tensorflow: An open source machine learning framework for everyone. `https://github.com/tensorflow/tensorflow`.

[3] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data, 2019.

[4] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le. Randaugment: Practical automated data augmentation with a reduced search space, 2019.

[5] R. Golemanova. The top 5 uses of image recognition, Dec 2019.

[6] D. Ho, E. Liang, I. Stoica, P. Abbeel, and X. Chen. Population based augmentation: Efficient learning of augmentation policy schedules, 2019.

[7] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features, 2019.

[8] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization, 2018.