Jointly Optimizing Preprocessing and Inference for DNN-based Visual Analytics

Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, Matei Zaharia
Stanford University

Abstract

In response to the growing computational cost of deep neural networks, researchers and companies have built accelerators and systems that have reduced the cost of executing DNN computational graphs by orders of magnitude. However, these impressive numbers ignore a key bottleneck in *end-to-end* DNN inference: the preprocessing of data and data transfer times. As we show, these costs are now over 3× the cost of DNN inference. To address the costs of preprocessing, we propose two techniques for optimizing across preprocessing and DNN execution. First, we optimize common visual DNN preprocessing operations. Second, we use low-resolution visual data for fast DNN inference. By using low-resolution data, we can trade off between accuracy and throughput *by just changing the input format*, sometimes even exceeding the accuracy of higher resolution DNNs. We implement these optimizations in a prototype DNN inference system, SMOL, and show that SMOL can achieve up to 4.2× improved throughput compared to standard DNNs.

1 Introduction

Deep neural networks have rapidly progressed in accuracy and breadth of tasks they can provide high accuracy on. As a result, these DNNs now power a range of visual analytics tasks and systems [1, 2]. *DNN execution*, or executing the computational graph of the DNN, can be computationally expensive, requiring up to many billions of floating point operations.

As a result, researchers have build hardware [3, 4], compilers [5, 6], and software systems [1, 2, 7], which has lead to orders of magnitude reduced cost. For example, the popular ResNet-50 model [8], which is widely used in DNN benchmarks [9, 10], can now execute 7.6k images/second (im/s) on the NVIDIA V100 GPU with TensorRT compared to 159 im/s on the K80 GPU.

While these speedups are impressive, performance measurements in this existing work largely ignores a key bottleneck in *end-to-end* DNN inference: preprocessing, or the process of reading, decoding, and transferring image data to DNN accelerators. Due to the advances in hardware accelerators and compilers, we show that preprocessing costs often *dominate end-to-end DNN inference*. For example, simply decoding JPEG compressed images achieves a throughput of only 1.6k images/s on a `n1-standard-8` Google Cloud Platform (GCP) instance; this instance type is the largest standard instance type that one V100 can be attached to. On the V100, this throughput is over 4× slower than the throughput of ResNet-50. Furthermore, matching the throughput of the V100 requires an equivalent power draw from the CPU, assuming perfect scaling.

As accelerators and systems become more efficient [4, 5], we argue that it will becoming increasingly critical to account for the *end-to-end* cost of DNN inference, crucially including the preprocessing and data transfer time. In light of these observations, we examine opportunities for more principled joint optimization between preprocessing and DNN execution. Specifically, the behavior of a DNN is closely coupled with its input format; rather than treating the input format as fixed, we consider methods of using transformed inputs in DNN inference.

To demonstrate the promise of optimizing across input format, preprocessing, and inference, we present two optimizations for end-to-end visual DNN inference. First, we optimize a large class of common visual DNN preprocessing operations. Second, we demonstrate how to use native low-resolution visual data, e.g., Instagram thumbnails, for fast DNN inference. We show that by simply using a different input format, we can trade off between accuracy and throughput more efficiently than standard methods of doing so. Furthermore, we show that naively implementing these optimizations can reduce accuracy and present a training procedure that can recover accuracy on low-resolution data.

We implement these optimizations in a prototype system, SMOL. These optimizations can give pareto improvements over naive strategies of DNN inference, with up to 4.2$\times$ improved throughput.

2 Optimizing Across Preprocessing and DNN Inference

2.1 Optimizing Preprocessing

We optimize preprocessing by re-ordering/pre-computing operations for reduced costs.

First, we can re-order resizing and computing a central crop or regions of interest (ROI), as many DNNs only require a portion of the image for inference. For JPEG images, each 8x8 block, or *macroblock*, in a JPEG image can be decoded independently [11]. Thus, when only a portion of the image is needed, e.g., for central cropping or when selecting an ROI, only the specified portion of the image need be decoded. SMOL will first find the smallest rectangle that aligns with the 8x8 macroblock border and contains the region. Then, SMOL will decode the rectangle and return the crop. We note that other image formats also have partial decoding options.

Second, a large class of DNN preprocessing steps consists of resizing an image, splitting the channels, and performing per-channel normalization. We can optimize these steps by reordering steps to reduce memory computation and pre-computing the result of the normalization. For example, we can resize before transforming to float, reducing memory accesses by 4$\times$.

2.2 Low-resolution Visual Data

Inference on low-resolution data. One key method SMOL leverages to optimize preprocessing is to use low-resolution visual data. Many services that serve images store low-resolution versions of the full-resolution data, either for previewing purposes or for low-bandwidth situation. For example, Instagram stores 161x161 previews of images at the time of writing [12].

To leverage low-resolution visual data, SMOL upscales the low-resolution images. However, naively upscaling gives low accuracy results. Instead, SMOL will train DNNs to be aware of low-resolution data. Furthermore, if several resolutions are available, SMOL will profile the accuracy and throughput of each one and selects the highest accuracy one for a fixed throughput (or vice versa).

Training for low-resolution data. As described, SMOL can use low-resolution visual data to decrease preprocessing costs. However, naively using low-resolution can decrease accuracy, especially for target DNNs. For example, using a standard ResNet-50 with native 161x161 images results in a *10.8% absolute drop in accuracy.* To alleviate the drop in accuracy, SMOL can train DNNs to be aware of low-resolution. This procedure can recover, or even exceed, the accuracy of standard DNNs. To accomplish this, SMOL explicitly augments images by downsampling them at training time.

We show that this training procedure can recover the accuracy of full resolution DNNs when using lossless low-resolution data, e.g., PNG compression. However, when using lossy low-resolution data, e.g., JPEG compression, low-resolution DNNs can suffer a drop in accuracy. Nonetheless, we show that using lossy low-resolution data can be more efficient than using smaller, full-resolution DNNs.

3 Evaluation

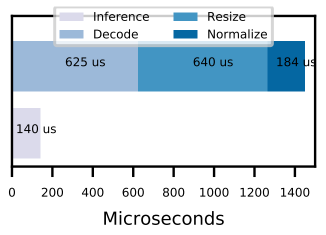
3.1 Experimental Setup

Dataset and models. We use the standard ResNets using the standard configurations as our baseline models, which takes in 224×224 images by default. We use the pretrained weights provided by the PyTorch `torchvision` package.

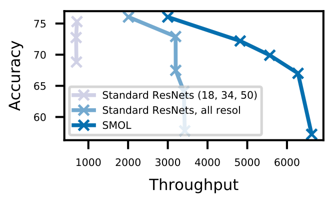
GPU	Release date	Throughput (ResNet-50)
K80	2014	159
P100	2016	1,955
V100	2017	7,151

Framework	Throughput (ResNet-50)
Keras	702
PyTorch	1,265
TensorRT	7,151

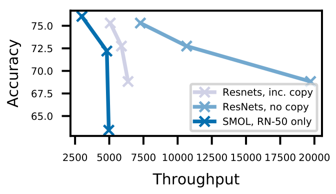
Table 1: **Right:** throughput of ResNet-50 on three GPU accelerators. Throughput has improved by over 44× in three years and will continue to improve. **Left:** Throughput of ResNet-50 on the V100 with three different execution environments. As shown, the efficient use of hardware can result in over a 10× improvement in throughput.



(a) Breakdown of end-to-end server of ResNet-50. DNN execution is bottlenecked by over 10× by preprocessing.



(b) Accuracy vs throughput for SMOL and baselines. Pareto-optimal frontier only.



(c) Accuracy vs throughput for SMOL and ResNet DNN execution only.

Figure 1: Accuracy vs throughput of SMOL, baselines, and DNN execution only. As shown, 1) SMOL can significantly outperform baselines, 2) SMOL can achieve within 1% of the theoretical throughput of the V100 for ResNet-50, 3) DNN execution can be bottlenecked by memory bandwidth.

We benchmark on the ImageNet dataset, which contains 50,000 JPEG compressed images in the test set. We encoded thumbnail versions of the ImageNet test set using: 1) JPEG $q = 75$, 2) JPEG $q = 95$, and 3) PNG compression all at a short size resolution of 161.

Hardware environment. Throughout, we use the Google Cloud Platform (GCP) `n1-standard-8` instance type with a single NVIDIA V100 GPU attached. The `n1-standard-8` has 8 vCPU cores with 30 GB of RAM. As vCPUs are hyperthreads, compute intensive workloads, such as image decoding, will achieve sublinear scaling compared to a single hyperthread. We chose the `n1-standard-8` instance type as it is the largest standard GCP instance type that a single V100 can be attached to. As we show, assuming perfect scaling, the power draw required for naive preprocessing is approximately the same as the power necessary for inference (approximately 250W each).

3.2 Bottlenecks in End-to-End Inference

We benchmark ResNet-50 in several configurations to see bottlenecks in end-to-end inference.

Effect of software and hardware on inference throughput. We benchmarked ResNet-50 on the K80, P100, and V100 GPUs to show the effect of improved accelerators on throughput. We used a batch size of 64. As shown in Table 1, throughput has improved by 44× in three years.

We benchmarked ResNet-50 throughput on the V100 GPU using Keras, PyTorch, and TensorRT [5] to show the effect of more efficient DNN compilers on throughput. As shown in Table 1, efficient use of accelerators via efficient compilers can result in up to a 10× improvement in throughput.

Breakdown of end-to-end DNN inference. DNN inference consists of steps beyond the execution of the DNN computation graph. For the standard ResNet-50 configuration, the preprocessing steps are: 1) decoding the compressed image, 2) resizing the image, 3) centrally cropping, 4) per-channel normalization, and 5) transpose to channels-first (this step depends on the network configuration).

To see the breakdown of preprocessing the costs, we implemented these steps in hand-optimized C++, ensuring best practices. We used `libturbo-jpeg`, an optimized library for JPEG decompression. We used OpenCV's optimized image processing libraries for the resize and normalization. We additionally did not time the time to load the data from disk. We use multithreading to utilize all the cores. As shown in Figure 1a, the preprocessing of the data achieves 10× lower throughput than ResNet-50.

3.3 SMOL can Improve Throughput

We show the efficacy of optimizing across preprocessing and inference by comparing the accuracy-throughput trade offs of using SMOL compared to the common practice of using a variety of models of different depths [8]. We plot the 1) standard ResNets (18, 34, and 50) using full resolution data, 2) standard ResNets using all available resolutions, and 3) ResNets trained with SMOL in Figure 1b. As shown, SMOL can substantially outperform both the standard ResNets with full resolution and reduced resolution images. Namely, SMOL is pareto optimal to these configurations.

To further investigate the source of speedups, we benchmark the following ablations: 1) only the accelerator costs, 2) the accelerator and data transfer costs, and 3) SMOL with ResNet-50 and only changing the input format. We show results in Figure 1c. We see that SMOL can achieve within 1% of the theoretical throughput of the V100 for ResNet-50. Further, we see that *using smaller models is bottlenecked by transfer times to the accelerator* for standard, cloud-configured instances. Thus, we see that changing the input format is an effective method of trading off accuracy and throughput.

4 Related Work

Visual analytics systems. Contemporary visual analytics systems leverage DNNs for high accuracy annotations and largely focus on optimizing the cost of executing these DNNs [1, 2, 13, 14, 7]. These systems typically use smaller proxy models, such as specialized NNs to accelerate analytics. However, as we have showed, modern hardware and compilers can create bottlenecks elsewhere in the end-to-end execution of DNNs, which will only be exacerbated by optimized software systems.

Optimizing DNN execution and DNN serving. Researchers have proposed compilers for optimizing DNN computation graphs, including TensorRT, XLA, and others [6, 5]. These compilers generally cannot optimize across preprocessing and inference. Furthermore, as they generate more efficient code, the bottleneck of preprocessing will only increase.

Optimizing DNN preprocessing. To the best of our knowledge, the only system that focuses on optimizing DNN preprocessing is NVIDIA DALI [15]. However, DALI optimizes preprocessing for DNN training, which is significantly slower than inference. While DALI is sufficient for training workloads, it does not match the throughput for inference.

5 Conclusion

In this work, we show that preprocessing data can be the bottleneck in end-to-end DNN inference. To alleviate this bottleneck, we introduce two techniques of optimizing across preprocessing and DNN execution. First, we optimize a large class of common visual DNN preprocessing operations and only decode the visual data necessary for inference. Second, we show how to use low-fidelity visual data for fast inference. Together, these optimizations can improve standard visual classification workloads, showing the promise of co-optimizing preprocessing and inference.

References

[1] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: optimizing neural network queries over video at scale. *PVLDB*, 10(11):1586–1597, 2017.

[2] Michael R Anderson, Michael Cafarella, Thomas F Wenisch, and German Ros. Predicate optimization for a visual analytics database. *ICDE*, 2019.

[3] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, pages 1–12. IEEE, 2017.

[4] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. A configurable cloud-scale dnn processor for real-time ai. In *ISCA*, pages 1–14. IEEE Press, 2018.

[5] Nvidia tensorrt, 2019.

[6] Chris Leary and Todd Wang. Xla: Tensorflow, compiled. *TensorFlow Dev Summit*, 2017.

[7] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. Accelerating machine learning inference with probabilistic predicates. In *SIGMOD*, pages 1493–1508. ACM, 2018.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[9] MLPerf. `https://mlperf.org/`, 2018.

[10] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition. *Training*, 100(101):102, 2017.

[11] Gregory K Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

[12] Elizabeth Arens. Always up-to-date guide to social media image sizes, 2019.

[13] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David Andersen, Michael Kaminsky, and Subramanya Dulloor. Scaling video analytics on constrained edge nodes. *SysML*, 2019.

[14] Daniel Kang, Peter Bailis, and Matei Zaharia. Challenges and opportunities in dnn-based video analytics: A demonstration of the blazeit video query engine. CIDR, 2019.

[15] NVIDIA. NVIDIA DALI, 2019.