Jointly Optimizing Preprocessing and Inference for DNN-based Visual Analytics

Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, Matei Zaharia
Stanford DAWN Project
ddkang@cs.stanford.edu,ankit96@stanford.edu,tejav@stanford.edu,pbailis@cs.stanford.edu,matei@cs.stanford.edu

ABSTRACT

While deep neural networks (DNNs) are an increasingly popular way to query large corpora of data, their significant runtime remains an active area of research. As a result, researchers have proposed systems and optimizations to reduce these costs by allowing users to trade off accuracy and speed. In this work, we examine *end-to-end* DNN execution in visual analytics systems on modern accelerators. Through a novel measurement study, we show that the *preprocessing of data* (e.g., decoding, resizing) can be the bottleneck in many visual analytics systems on modern hardware.

To address the bottleneck of preprocessing, we introduce two optimizations for *end-to-end* visual analytics systems. First, we introduce novel methods of achieving accuracy and throughput trade-offs by using natively present, low-resolution visual data. Second, we develop a runtime engine for efficient visual DNN inference. This runtime engine a) efficiently pipelines preprocessing and DNN execution for inference, b) places preprocessing operations on the CPU or GPU in a hardware- and input-aware manner, and c) efficiently manages memory and threading for high throughput execution. We implement these optimizations in a novel system, Smol, and evaluate Smol on eight visual datasets. We show that its optimizations can achieve up to 5.9× *end-to-end* throughput improvements at a fixed accuracy over recent work in visual analytics.

PVLDB Reference Format:
Daniel Kang, Ankit Mathur, Teja Veeramacheneni,
Peter Bailis, Matei Zaharia . Jointly Optimizing Preprocessing and Inference for DNN-based Visual Analytics. PVLDB, 14(2): 87 - 100, 2021.
doi:10.14778/3425879.3425881

1 INTRODUCTION

Deep neural networks (NNs) now power a range of visual analytics tasks and systems [6, 31, 34, 36] due to their high accuracy, but state-of-the-art DNNs can be computationally expensive. For example, accurate object detection methods can execute as slow as 3-5 frames per second (fps) [27, 62].

To execute visual analytics queries efficiently, systems builders have developed optimizations to trade off accuracy and throughput [6, 31, 34, 36, 42]: more accurate DNNs are more computationally expensive [28, 61, 62]. Many of these systems (e.g., NoScope, BlazeIt, Tahoma, and probablistic predicates) accelerate visual analytics queries by using proxy or *specialized NNs*, which approximate larger target DNNs. These specialized NNs can be up to 5 orders of magnitude cheaper to execute than their target DNNs and are used to filter inputs so the target DNNs will be executed fewer times [6, 31, 34, 36, 42].

This prior work focuses solely on reducing DNN execution time. These systems were built before recent DNN accelerators were introduced and were thus benchmarked on older accelerators. In this context, these systems correctly assume that DNN execution time is the overwhelming bottleneck. For example, Tahoma benchmarks on the NVIDIA K80 GPU, which executes ResNet-50 (a historically expensive DNN [1, 17, 18]) at 159 images/second.

However, as accelerators and compilers have advanced, these systems ignore a key bottleneck in *end-to-end* DNN inference: preprocessing, or the process of decoding, transforming, and transferring image data to accelerators. In the first measurement study of its kind, we show that preprocessing costs often *dominate end-to-end DNN inference* when using advances in hardware accelerators and compilers. For example, the historically expensive ResNet-50 [1, 18] has improved in throughput by 28× on the inference-optimized NVIDIA T4 GPU. As a result, ResNet-50 is now 9× higher throughput than CPU-based image preprocessing, making *preprocessing the bottleneck*, on the inference-optimized `g4dn.xlarge` Amazon Web Services (AWS) instance, which has a NVIDIA T4. This boost in efficiency translates to both power and dollar costs: preprocessing requires approximately 2.3× as much power and costs 11× as much as DNN execution (§7). Similar results hold for Google Cloud's T4 inference optimized instances. These imbalances become only higher with smaller specialized NNs that recent visual analytics systems use.

In light of these observations, we examine opportunities for more principled *joint optimization* of preprocessing and DNN execution, especially for preprocessing-bound, high-throughput batch analytics workloads. We leverage two insights: a) the accuracy and throughput of a DNN is closely coupled with its input format and b) preprocessing operations can be placed on both CPUs and accelerators. Thus, rather than treating the input format as fixed, we consider methods of using inputs as a key step in DNN architecture search and training.

This yields two novel opportunities for accelerating inference: a) cost-based methods that leverage low-resolution visual data for higher accuracy or improved throughput and b) input- and hardware-aware methods of placing preprocessing operations on the CPU or accelerator and correctly pipelining computation.

A critical component to leverage these opportunities is a cost model to select query plans. We correct the erroneous assumption in prior work that DNN execution dominates *end-to-end* DNN inference. We instead propose a cost model that is preprocessing aware and validate that our cost model is more accurate than prior

Proceedings of the VLDB Endowment, Vol. 14, No. 2 ISSN 2150-8097.
doi:10.14778/3425879.3425881

cost models. While our preprocessing aware cost model is simple, it enables downstream optimizations, described below.

First, we propose methods of using natively present, low resolution visual data for more efficient, input-aware accuracy/throughput trade offs. Image and video serving sites often have natively present low resolution data, e.g., Instagram has thumbnails [7] and YouTube stores multiple resolutions of the same video. Even when low resolution data is not natively present, we can partially decode visual data (e.g., omitting the deblocking filter in H.264 decoding). As such, we can use natively present data or partial decoding for reduced preprocessing costs. However, naively using this reduced fidelity data can reduce accuracy. To recover accuracy, we propose an augmented DNN training procedure that explicitly uses data augmentation for the target resolution. Furthermore, we show that using larger, more accurate DNNs on low resolution data can result in higher accuracy than smaller DNNs on full resolution data. Enabled by our new preprocessing-aware cost model, we can select input formats and DNN combinations that achieve better accuracy/throughput trade offs.

Second, we decide to place preprocessing operations on the CPU or accelerator to balance the throughput of DNN execution and preprocessing. Furthermore, to enable high-performance pipelined execution, we build an optimized runtime engine for end-to-end visual DNN inference. Our optimized runtime engine makes careful use of pipelined execution, memory management, and high-performance threading to fully utilize available hardware resources.

We implement these optimizations in Smol, a runtime engine for end-to-end DNN inference that can be integrated into existing visual analytics systems. We use Smol to implement the query processing methods of two modern visual analytics systems, BlazeIt [34] and Tahoma [6], and evaluate Smol on eight visual datasets, including video and image datasets. We verify our cost modeling choices through benchmarks on the public cloud and show that Smol can achieve up to 5.9× improved throughput on recent GPU hardware compared to recent work in visual analytics.

In summary, we make the following contributions:

(1) We show that preprocessing costs can dominate end-to-end DNN-based analytics when carefully using modern hardware.
(2) We illustrate how to use natively-encoded low-resolution visual data formats and specialized NNs to achieve input-aware accuracy/throughput trade-offs.
(3) We propose and implement methods of further balancing preprocessing and DNN execution, including hardware- and input-aware placement of preprocessing operations.

2 MEASUREMENT STUDY OF END-TO-END DNN INFERENCE

We benchmark DNNs and visual data preprocessing on the public cloud, showing that *preprocessing costs* can now dominate end-to-end DNN inference. We show that these trends arise from dramatically improved new accelerators reducing dollar and power costs of DNN execution, and efficient use of hardware.

We benchmark throughputs on the inference-optimized T4 GPU with a dollar cost-balanced number of vCPU cores on an AWS instance. Our benchmarks show that preprocessing dominates in both dollar cost and power costs. For example, preprocessing requires 2.2× as much power (158W vs 70W) and costs 11× as much for ResNet-50 ($2.37 vs $0.218, §7). These trends are similar for other cloud providers (e.g., Google Cloud Platform's T4-attachable instances and Microsoft Azure's newly announce T4 instances) and instance types.

Execution environment	Throughput (im/s)
Keras	243
PyTorch	424
TensorRT	4,513

Table 1: Throughput of ResNet-50 on the T4 with three different execution environments. Keras was used in [6]. The efficient use of hardware can result in over a 17× improvement in throughput. We used the optimal batch size for each framework (64, 256, and 64 respectively).

Experimental setup. We benchmarked the popular ResNet-50 model for image classification [28], which has widely been used in benchmarking [1, 18] and has been considered expensive. Specialized NNs are typically much cheaper than ResNet-50.

We benchmarked the time for only DNN execution and the time for preprocessing separately to isolate bottlenecks.

We benchmarked on the publicly available inference-optimized NVIDIA T4 GPU [46]. We used the `g4dn.xlarge` AWS instance which has 4 vCPU cores (hyperthreads): this configuration is cost balanced between vCPUs and the accelerator (§7). This instance type is optimized for DNN inference; similar instances are available on other cloud providers. We used the TensorRT compiler [2] for optimized execution. While we benchmarked on the T4, other contemporary, non-public accelerators report similar or improved results [22, 32].

Effect of software on throughput. We benchmarked ResNet-50 throughput on the inference-optimized T4 GPU using three software systems for DNNs to show how more efficient software affects throughput. We benchmark using Keras [16], PyTorch [50], and TensorRT [2]. We note that Keras was used by Tahoma and TensorRT is an optimized DNN computational graph compiler.

As shown in Table 1, efficient use of accelerators via optimized compilers (TensorRT) can result in up to a 10× improvement in throughput. Importantly, preprocessing becomes the bottleneck with the efficient use of accelerators.

Breakdown of end-to-end DNN inference. DNN inference includes preprocessing. For the standard ResNet-50 configuration, the preprocessing steps are [28, 43]:

(1) Decode the compressed image, e.g., JPEG compressed.
(2) Resize the image with an aspect-preserving resize such that the short edge of the image is 256 pixels. Centrally crop the image to 224x224.
(3) Convert the image to float32. Divide the pixel values by 255, subtract a per-channel value, and divide by a per-channel value (these values are derived from the training set).
(4) Rearrange the pixel values to channels-first (this step depends on the DNN configuration).

To see the breakdown of preprocessing the costs, we implemented these preprocessing steps in hand-optimized C++, ensuring best practices for high performance C++, including reuse of memory to avoid allocations. We used `libturbo-jpeg`, a highly optimized library for JPEG decompression, for decoding the JPEG images. We used OpenCV's optimized image processing libraries for the resize

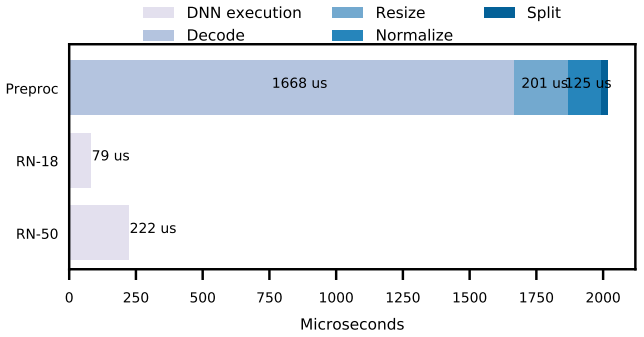

Figure 1: Breakdown of end-to-end inference per image of ResNet-50 and 18 for a batch size of 64 on the inference-optimized AWS `g4dn.xlarge` instance type (one NVIDIA T4 GPU and 4 vCPU cores). The DNN was executed on the T4 and the preprocessing was parallelized across all CPU cores. The execution of the DNN is 7.1× and 22.9× faster than preprocessing data for ResNet-50 and 18 respectively.

and normalization. For DNN execution, we executed the DNNs with TensorRT and multiple CUDA streams on synthetic images. We run all benchmarks on a standard `g4dn.xlarge` AWS instance and use multithreading to utilize all the cores.

As shown in Figure 1, simply decoding the JPEG files achieves lower throughput than the throughput of ResNet-50 execution. All together, the preprocessing of the data achieves 7.1× lower throughput than ResNet-50 execution. These overheads increase to up to 22.9× for ResNet-18. As discussed, preprocessing dominates in terms of power and dollar costs as well.

Similar results hold for other networks, such as the MobileNet-SSD [30, 41] used by MLPerf Inference [57]. This DNN executes at 7,431 im/s, compared to a preprocessing throughput of 397 im/s on the MS-COCO dataset.

Discussion. Several state-of-the-art DNNs execute far slower than the DNNs benchmarked in this section, e.g., a large Mask R-CNN may execute at 3-5 fps. However, many systems use specialized NNs to reduce invocations of these large DNNs. For example, the BlazeIt system uses a specialized NN to approximate the larger DNN, which reduces the number of large DNN invocations [34]. As these specialized NNs are small (potentially much smaller than even ResNet-50), we believe our benchmarks are of wide applicability to DNN-based visual analytics.

3 SMOL OVERVIEW

To reduce the imbalance between preprocessing and DNN execution, we develop a novel system, Smol. Smol's goal is to execute *end-to-end* batch visual analytics queries. Unlike prior work, Smol aims to optimize end-to-end query time, including the computational cost of preprocessing in addition to the computational cost of DNN execution.

To execute these visual analytics queries, Smol uses a cost-based model to generate query plans that span preprocessing and DNN execution. Smol executes these plans in its optimized end-to-end inference engine. For a given query system (e.g., Tahoma or BlazeIt), Smol's cost model must be integrated into the system.

We show a schematic of Smol's architecture in Figure 2.

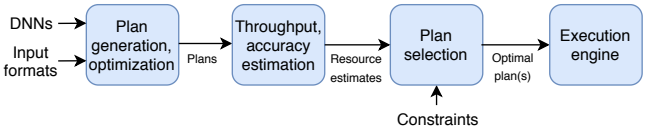

Figure 2: System diagram of Smol. As input, Smol takes a set of DNNs, visual input formats, and optional constraints. As output, Smol returns an optimal set of plans or plan, depending on the constraints. Smol will generate plans, estimate the resources for each plan, and select the Pareto optimal set of plans.

Model	Throughput	Preproc. bound?
MobileNet-V1	16,885	Yes
VGG19	1,889	Yes
Inception V4	1,339	Yes
ResNet-50	4,513	Yes
ResNeXt-101	1,724	Yes
SSD MobileNet-V1 (300)	7,609	Yes
SSD ResNet (1200)	137	No
Mask R-CNN (1200)	14	No

Table 2: Throughput of various models on the T4 GPU (classification models on the top and detection models on the bottom) [68]. As shown, all but the largest, state-of-the-art detection models are preprocessing bound.

3.1 System Overview

Deployment setting. In this work, we focus on high throughput batch settings, as recent work does [6, 34, 37, 70]. Smol's goal is to achieve the highest throughput on the available hardware resources. For example, a visual analytics engine might ingest images or videos daily and run a batch analytics job each night. Smol is most helpful for preprocessing-bound workloads (Table 2). As we describe, Smol accepts models exported from training frameworks (e.g., PyTorch, TensorFlow, or Keras) and optimizes its inference. As such, it is designed to be used at inference time, not with training frameworks.

Nonetheless, several of our techniques, particularly in jointly optimizing preprocessing and inference, also apply to the low-latency or latency-constrained throughput settings.

We note that in high throughput batch settings, visual data is almost always stored in compressed formats that require preprocessing. Uncompressed visual data is large: a single hour of 720p video is almost 900GB of data whereas compressed video can be as small as 0.5GB per hour. Similarly, JPEG images can be up to 10× smaller than uncompressed still images.

Smol inference. As inputs, Smol will take a set of trained DNNs and a set of natively available visual data formats (e.g., full resolution JPEG images, thumbnail JPEGs). We denote the set of DNNs as $\mathcal{D}$ and the set of visual data formats as $\mathcal{F}$. Smol further takes a set of calibration images (i.e., a validation set) to estimate accuracy.

Given these inputs, Smol will estimate costs to select a plan (concretely, a DNN and an input format). Smol will then optimize this plan and execute it.

Smol optionally takes a throughput or accuracy constraint at inference time. If a constraint is specified, Smol will select an optimized execution plan that respects these constraints. Otherwise,

Smol will execute the highest throughput plan. Smol can be integrated with other systems by returning a Pareto optimal set of plans (in accuracy and throughput). The calling system will then select a plan that Smol will execute.

Smol training. While the user can provide the set of trained DNNs, Smol can optionally train specialized NNs as well. Given a set of DNN architectures (e.g., ResNets) and the natively available formats, Smol will choose to train some or all of the DNNs. Given the initial set of models on full resolution data, Smol will fine-tune the networks on the cross product of $\mathcal{D}$ and resolutions (Smol will use the same model for different formats of the same resolution). As Smol fine-tunes, this process adds at most a 30% overhead in training in the settings we consider. Smol can also train these network at execution time [34].

Components. Smol implements the training phase as other systems do [6, 34, 36]. As training specialized NNs has been studied in depth in prior work, we defer discussion to this prior work. Smol differs from these systems only in it's low-resolution augmented training (discussed below).

At inference time, Smol contains three major components: 1) a plan generator, 2) a cost estimator, and 3) an execution engine. We show these components in Figure 2.

Smol first generates query plans from $\mathcal{D}$ and $\mathcal{F}$ by taking $\mathcal{D}\times\mathcal{F}$. For each plan, Smol will estimate the relative costs of preprocessing and DNN execution and decide where to place preprocessing operations (i.e., on the CPU or accelerator) for highest throughput. Given these optimized plans, Smol will estimate the accuracy and throughput of these plans using its cost model. This process is cheap compared to training, so Smol exhaustively benchmarks the Pareto frontier of $\mathcal{D}\times\mathcal{F}$. Smol uses a preprocessing-aware cost model, in contrast to prior work that ignores these costs. Finally, Smol will return the best query plan if a constraint is specified or the Pareto optimal set of query plans if not.

Optimizations. To efficiently execute queries, Smol has several optimizations for improved accuracy/throughput trade offs and an efficient DNN execution engine.

Briefly, Smol achieves improved accuracy and throughput trade offs by considering an expanded set of DNNs and leveraging natively present low-resolution data (§5). In contrast, prior work considers only one input format. From the selected DNN and input format, Smol will efficiently execute such plans by placing preprocessing operations on CPUs or accelerators in a hardware- and input-aware manner, efficiently pipelining computation stages, and optimizing common preprocessing operations (§6). We describe these optimizations in detail below.

3.2 Examples

Classification example. Smol can be incorporated into prior work that uses specialization for classification queries [6, 36, 42]. These queries are often binary classification queries, e.g., the presence or absence of a car in a video. We describe Tahoma in this example, but note that other systems are similar in spirit.

Tahoma uses a fixed target model and considers a fixed input format, namely the provided input format of full-resolution JPEG images. Tahoma considers 24 specialized NN, each of which are cascaded with the target DNN. Thus, $|\mathcal{F}|=1$ and $|\mathcal{D}|=24$. Tahoma aims to return the configuration with the highest throughput for a given accuracy. Tahoma estimates the throughput of $D_i \in \mathcal{D}$ by *adding* preprocessing costs, which we show leads to inaccurate throughput estimates. We further note that Tahoma considers downsampling full resolution images for improved DNN execution, but not for reduced preprocessing costs.

ResNet	Throughput	Accuracy
ResNet-18	12,592	68.2%
ResNet-34	6,860	71.9%
ResNet-50	4,513	74.34%

Table 3: Throughput and top-one accuracy for ResNets of different depths. As shown, there is a trade off between accuracy and throughput (i.e., computation).

In contrast, Smol can use natively present thumbnail images, which would expand $\mathcal{F}$. Decoding these thumbnail images is significantly more efficient, resulting in higher throughput.

Aggregation example. Smol can be incorporated into prior work that uses specialized NNs for aggregation queries over visual data, e.g., the number of cars in a video. The recent BlazeIt system uses specialized NNs as a control variate to reduce the variance in sampling [34]. As the variance is reduced, this procedure results in fewer target model invocations compared to standard random sampling. BlazeIt trains a single specialized NN ($|\mathcal{D}|=1$) and uses a fixed input format ($|\mathcal{F}|=1$).

In contrast, Smol can use an expanded set of videos which are encoded at different resolutions. Namely, Smol considers $|\mathcal{F}|>1$. These other formats are natively present in many serving applications, e.g., for thumbnail or reduced bandwidth purposes.

We describe Smol's cost model, optimizations, its implementation, and its evaluation below.

4 COST MODELING FOR VISUAL ANALYTICS

When deploying DNN-based visual analytics systems, application developers have different resource constraints. As such, these systems often expose a way of trading off between accuracy and throughput. Higher accuracy DNNs typically require more computation: we demonstrate this property on the popular ImageNet dataset [19] with standard ResNets in Table 3. Prior work has designed high throughput specialized DNNs for filtering [6, 34, 36]. We do not focus on the design of DNNs in this work and instead use standard DNNs (§5).

One popular method for DNN selection is to use a cost model [6, 36]. We describe cost modeling for DNNs and how prior work estimated the throughput of DNN execution. Critically, these prior cost models ignore preprocessing costs or ignore that preprocessing can be pipelined with DNN execution. We show that ignoring these factors can lead to inaccurate throughput estimations (Table 4). We then describe how to make cost models preprocessing-aware.

Cost models. Given resource constraints and metrics to optimize, a system must choose which DNNs to deploy to maximize these metrics while respecting constraints. For example, one popular resource constraint is a minimum throughput and one popular metric is accuracy. As such, we focus on throughput-constrained accuracy and accuracy-constrained throughput.

Config.	Preprocessing throughput (im/s)	DNN execution throughput (im/s)	Pipelined throughput (im/s)	Smol estimate (% error, estimate)	BlazeIt estimate (% error, estimate)	Tahoma estimate (% error, estimate)
Balanced	4001	4999	4056	**1.4%**, 4001	23.2%, 4999	44.8%, 2222
Preproc-bound	534	4999	557	**4.1%**, 534	797.5%, 4999	9.3%, 482
DNN-bound	5876	1844	1720	**7.2%**, 1844	**7.2%**, 1884	22.7%, 1403

Table 4: We show measurements of preprocessing, DNN execution, and pipelined end-to-end DNN inference for three configurations of DNNs and input formats: balanced, preprocessing-bound, and DNN-execution bound. We measure the throughput in images per second of preprocessing, DNN execution, and end-to-end DNN inference on the left. We show the throughput estimation and error in estimation for three cost models on the right. We bold the most accurate estimate. As shown, Smol matches or ties the most accurate estimate for all conditions.[1]

Specifically, denote the possible set of system configurations as $C_1, ..., C_n$. Denote the resource consumption estimate of each configuration as $R(C)$ and the resource constraint as $R_{\max}$. Denote the metric to optimize as $M(C)$.

In its full generality, the optimization problem is

$$\max_i M(C_i) \quad \text{s.t. } R(C_i) \leq R_{\max}. \tag{1}$$

In this framework, both accuracy and throughput can either be constraints or metrics. For example, for throughput-constrained accuracy, $R(C_i)$ would be an estimate of the throughput of C_i and $M(C_i)$ would be an estimate of the accuracy of C_i. Similarly, for accuracy-constrained throughput, $R(C_i)$ would be an estimate of the accuracy and $M(C_i)$ would be an estimate of the throughput.

As an example, Tahoma generates $C_i = [D_{i,1}, ..., D_{i,k}]$ to be a sequence of k models, $D_{i,j}$, that are executed in sequence. The resource $R(C_i) = A(C_i)$ is the accuracy of configuration C_i and the metric $M(C_i) = T(C_i)$ is the throughput of configuration C_i.

Prior work has focused on expanding the set of C_i or evaluating $R(C_i)$ and $M(C_i)$ efficiently [6, 10, 36, 42]. A common technique is to use a smaller model (e.g., a specialized NN) to filter data before executing a larger, target DNN in a *cascade*. For example, when detecting cars in a video, NoScope will train an efficient model to filter out frames without cars [36]. Cascades can significantly expand the feasible set of configurations.

For cost models to be effective, the accuracy and throughput measurements must be accurate. We discuss throughput estimation below. Accuracy can be estimated using best practices from statistics and machine learning. A popular method is to use a held-out validation set to estimate the accuracy [8]. Under the assumption that the test set is from the same distribution as the validation set, this procedure will give an estimate of the accuracy on the test set.

Throughput estimation. A critical component of cost model for DNNs is the throughput estimation of a given system configuration C_i; recall that C_i is represented as a sequence of one or more DNNs, $D_{i,j}$. Given a specific DNN $D_{i,j}$, estimating its throughput simply corresponds to executing the computation graph on the accelerator and measuring its throughput. As DNN computation graphs are typically fixed, this process is efficient and accurate.

Estimation ignoring preprocessing. Prior work (e.g., probablistic predicates, BlazeIt, NoScope) [34, 36, 42] has used the throughput of $D_{i,j}$ to estimate the throughput of end-to-end DNN inference. Specifically, BlazeIt and NoScope estimates the throughput, $\hat{T}(C_i)$ as

$$\hat{T}(C_i) \approx \frac{1}{\sum_{j=1}^{k} \frac{1}{\alpha_j^{-1} T_{\text{exec}}(D_{i,j})}} \tag{2}$$

where α_j is the pass-through rate of DNN $D_{i,j}$ and $T_{\text{exec}}(D_{i,j})$ is the throughput of executing $D_{i,j}$. $T_{\text{exec}}(D_{i,j})$ can be directly measured using synthetic data and α_j can be estimated with a validation set. This approximation holds when the cost of preprocessing is small compared to the cost of executing the DNNs.

However, this cost model ignores preprocessing costs. As a result, it is inaccurate when preprocessing costs dominate DNN execution costs or when preprocessing costs are approximately balanced with DNN execution costs (Table 4).

Estimation ignoring pipelining. Other systems (e.g., Tahoma) [6] estimate end-to-end DNN inference throughput as

$$\hat{T}(C_i) \approx \frac{1}{\frac{1}{T_{\text{preproc}}(C_i)} + \frac{1}{T_{\text{exec}}(C_i)}}. \tag{3}$$

This approximation ignores that preprocessing can be pipelined with DNN execution. As a result, this approximation holds when either preprocessing or DNN execution is the overwhelming bottleneck, but is inaccurate for other conditions, namely when preprocessing costs are approximately balanced with DNN execution costs (Table 4).

These throughput approximations (that ignore preprocessing costs and ignore pipelining) ignore two critical factors: 1) that input preprocessing can dominate inference times and 2) that input preprocessing can be pipelined with DNN execution on accelerators. We now describe a more accurate throughput estimation scheme.

Corrected throughput estimation. For high throughput DNN inference on accelerators, the DNN execution and preprocessing of data can be pipelined. As a result, Smol uses a more accurate throughput estimate for a given configuration:

$$\hat{T}(C_i) \approx \min\left(T_{\text{preproc}}(C_i), \frac{1}{\sum_{j=1}^{k} \frac{1}{\alpha_j^{-1} T_{\text{exec}}(D_{i,j})}}\right) \tag{4}$$

Importantly, preprocessing can dominate end-to-end DNN inference (§2). While there are some overheads in pipelining computation, we empirically verify the min approximation (§8.3).

If preprocessing costs are fixed, then it becomes optimal to maximize the accuracy of the DNN subject to the preprocessing throughput. Namely, the goal is to pipeline the computation as effectively as possible. We give two examples of how this can change which configuration is chosen.

First, when correctly accounting for preprocessing costs in a throughput-constrained accuracy deployment, it is not useful

[1] Preprocessing having lower throughput than both in the preprocessing-bound and balanced conditions are due to the experimental harness being optimized for pipelined execution. The experimental harness does not significantly affect throughput when compared to without the harness.

to select a throughput constraint higher than the throughput of preprocessing. Second, for an accuracy-constrained throughput deployment, the most accurate DNN subject to the preprocessing throughput should be selected.

5 INPUT-AWARE METHODS FOR ACCURACY AND THROUGHPUT TRADE OFFS

Given the corrected cost model, Smol's goal is to maximize the minimum of the preprocessing and DNN execution throughputs. However, if the input format and resolution are fixed, preprocessing throughputs are fixed and can be lower than DNN execution throughputs. To provide better accuracy and throughput trade-offs, we propose three techniques: 1) expanding the search space of specialized DNNs, 2) using natively present, low resolution visual data, and 3) a DNN training technique to recover accuracy loss from naively using low resolution visual data.

5.1 Expanding search space

As described, many systems only consider cheap, specialized NNs. Concretely, BlazeIt and Tahoma considers specialized NNs that can execute up to 250,000 images/second, which far exceeds preprocessing throughputs for standard image and video encodings. As DNNs are generally more accurate as they become more expensive, these systems use specialized NNs that are less accurate relative to preprocessing throughput-matched NNs.

In contrast, Smol considers NNs that have been historically considered expensive. We have found that standard ResNet configurations [28] (18 to 152) strongly outperforms specialized NNs used in prior work. Furthermore, ResNet-18 can execute at 12.6k images/second, which generally exceeds the throughput of preprocessing. Thus, Smol currently uses these ResNets as the specialized NNs. As hardware advances, other architectures (e.g., ResNeXt [69]) may be appropriate.

5.2 Low-resolution data

Overview. Many visual data services store the data at a range of resolutions. Low-resolution visual data is typically stored for previewing purposes or for low-bandwidth situations. For example, Instagram stores 161x161 previews of images [7]. Similarly, YouTube stores several resolutions of the same video for different bandwidth requirements, e.g., 240p up to 4K video.

Decoding low-resolution visual data is more efficient than decoding full resolution data. Smol could decode and then upscale the low-resolution visual data for improved preprocessing throughput. However, we show that naively upscaling gives low accuracy results. Instead, Smol will train DNNs to be aware of low-resolution data, as described below (§5.3).

Recent work uses lower resolution data to improve NN throughput, *but not to reduce preprocessing costs* [6, 71]. These systems decode full-resolution data and downsamples the data, which does not improve preprocessing throughput.

Selecting DNNs and resolution jointly. Many systems provide accuracy and throughput trade-offs by cascading a specialized NN and a more accurate, target DNN [6, 36, 42]. However, these specialized NNs are often bottlenecked by preprocessing costs.

Instead, Smol uses low resolution data reduce preprocessing costs, and therefore end-to-end execution costs. However, low resolution visual data discards visual information and can result in lower accuracy in many cases. Nonetheless, Smol can provide accuracy and throughput trade-offs by carefully selecting DNN and input format combinations.

As a motivating example, consider ResNet-34 and 50 as the DNNs, and full resolution and 161x161 PNG thumbnails as the input formats. ResNet-34 and ResNet-50 execute at 6,861 and 4,513 images/second. On full resolution data, they achieve 72.72% and 75.16% accuracy on ImageNet, respectively. On low resolution data, they achieve 72.50% and 75.00% accuracy, respectively (when upscaling the inputs to 224x224 and using Smol's augmented training procedure). Full resolution and 161x161 thumbnails decode at 527 and 1,995 images/second, respectively. In this example, executing ResNet-50 on 161x161 thumbnails outperforms executing ResNet-34 on full resolution data, as end-to-end execution is bottlenecked by preprocessing costs.

Thus, Smol jointly considers both the input resolution format and the DNN. For classification, Smol also considers using a *single* DNN for accuracy/throughput trade-offs, instead of cascading a specialized DNN and target DNN.

For a given input format, Smol will only consider DNNs that exceed the throughput of the preprocessing costs and select the highest accuracy DNN subject to this constraint. As we have demonstrated, in certain cases, this will result in selecting lower resolution data with more expensive DNNs, contrary to prior work.

5.3 Training DNNs for Low-resolution

As described above, 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 upscaled to the standard 224x224 input resolution results in a *10.8% absolute drop in accuracy*. This drop in accuracy is larger than switching from a ResNet-50 to a ResNet-18, i.e., nearly reducing the depth by a third. 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.

Smol trains DNNs to be aware of low-resolution by augmenting the input data at training time. At training time, Smol will downsample the full-resolution inputs to the desired resolution and then upsample them to the DNN input resolution. Smol will do this augmentation in addition to standard data augmentation. By purposefully introducing downsampling artifacts, these DNNs can be trained to recover high accuracy on low-resolution data.

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.

6 AN OPTIMIZED RUNTIME ENGINE FOR END-TO-END VISUAL INFERENCE

In order to efficiently execute *end-to-end* visual inference in the high-throughput setting, we must make proper use of all available

hardware. We describe how to efficiently pipeline preprocessing and DNN execution for full use of hardware resources, how to optimize common preprocessing operations, how to place operations on CPUs or accelerators, and methods of partially decoding visual data. Several of these optimizations have been explored in other contexts, but not for end-to-end DNN inference [20, 23].

6.1 Efficient Use of Hardware

In order to efficiently use all available hardware resources, Smol must efficiently pipeline computation, use threads, and use/reuse memory.

As executing DNNs requires computation on the CPU and accelerator, Smol must overlap the computation. To do this, Smol uses a multi-producer, multi-consumer (MPMC) queuing system to allow for multithreading. The producers decode the visual data and the consumers perform DNN execution. Smol uses multiple consumers to leverage multiple CUDA streams. As preprocessing is data parallel and issuing CUDA kernels is low overhead, we find that setting the number of producers to be equal to the number of vCPU cores to be an efficient heuristic for non-NUMA servers.

An important performance optimization to effectively use the MPMC queuing system is reusing memory and efficient copying to the accelerator. Prior work that focuses on efficient preprocessing for training must pass memory buffers that contain the preprocessed images to the caller, which does not allow for efficient memory reuse. In contrast, the caller to Smol only requires the result of inference, not the intermediate preprocessed buffer. As a result, Smol can reuse these buffers. Furthermore, accelerators require pinned memory for efficient memory transfer. Reusing pinned memory results in substantially improved performance. Smol will further over-allocate memory to ensure that producer threads will not contend on consumers.

6.2 Optimizing Preprocessing Operations

A large class of common visual DNN preprocessing operations fall under the steps described in §2. Briefly, they include resizing, cropping, pixel-level normalization, data type conversion, and channel reordering. We can optimize these operations at inference time by fusing, reordering, and pre-computing operations.

To optimize these steps, Smol will accept the preprocessing steps as a computation directed, acyclic graph (DAG) and performs a combination of rule-based and cost-based optimization of these steps. To optimize a computation DAG, Smol will exhaustively generate possible execution plans, apply rule-based optimization to filter out plans, and perform cost-based optimization to select between the remaining plans.

Smol contains rules of allowed operation reordering to generate the possible set of execution plans:

(1) Normalization and data type conversion can be placed at any point in the computation graph.
(2) Normalization, data type conversion, and channel reordering can be fused.
(3) Resizing and cropping can be swapped.

Once Smol generates all possible execution plans, Smol will then apply the following rules to prune plans:

(1) Resizing is cheaper with fewer pixels.
(2) Resizing is cheaper with smaller data types (e.g., `INT8` resizing is cheaper than `FLOAT32` resizing).
(3) Fusion always improves performance.

We currently implement fusion manually, but code generation could also be applied to generate these kernels [47]. Given a set of plans after rule-based pruning, Smol approximates the cost by counting the number of arithmetic operations in each plan for the given data types. Smol will select the cheapest plan.

6.3 Preprocessing Operator Placement

In addition to optimizing common preprocessing operations, Smol can place preprocessing operations on the CPU or accelerator. Depending on the input format/resolution and DNN, the relative costs of preprocessing and DNN execution may differ. For example, small specialized NNs may execute many times faster than preprocessing, but a state-of-the-art Mask R-CNN may execute slower than preprocessing.

As a result, to balance preprocessing and DNN execution costs, it may be beneficial to place operations on either the CPU or accelerator. Furthermore, many preprocessing operations (e.g., resizing, normalization) are efficient on accelerators, as the computational patterns are similar to common DNN operations.

If DNN execution dominates, then Smol will place as many operations on the CPU as possible, to balance costs. If preprocessing cost dominate, then Smol will place as many operations on the accelerator as possible. Since preprocessing operations are sequential, Smol need only consider a small number (typically under 5) configurations for a given model and image format.

6.4 Partial and Low-Fidelity Decoding

Overview of Visual Compression Formats. We briefly describe salient properties of the majority of popular visual compression formats, including the popular JPEG, HEVC/HEIC, and H.264 compression formats. We describe the decoding of the data and defer a description of encoding to other texts [52, 60, 67].

Decoding generally follows three steps: 1) entropy decoding, 2) inverse transform (typically DCT-based), and 3) optional post-processing for improved visual fidelity (e.g., deblocking).

Importantly, the entropy decoders in both JPEG and HEVC (Huffman decoding and arithmetic decoding respectively) are not efficient on accelerators for DNNs as it requires substantial branching. Furthermore, certain parts of decoding can be omitted, e.g., the deblocking filter, for reduced fidelity but faster decoding times.

Leveraging partial decoding. When low-resolution visual data is not available, Smol optimizes preprocessing by partially decoding visual data. Many DNNs only require a portion of the image for inference, or *regions of interest (ROI)*. For example, many image classification networks centrally crop images, so the ROI is the central crop. Computing face embeddings crops faces from the images, so the ROIs are the face crops. Furthermore, these networks often take standard image sizes, e.g., 224×224. We show two examples in Figure 3. Computing ROIs may require expensive upstreaming processing in some applications, e.g., executing a detection DNN.

Many image compression formats allow for partial decoding explicitly in the compression standard and all compression formats we are aware of allow for early stopping of decoding. We give three

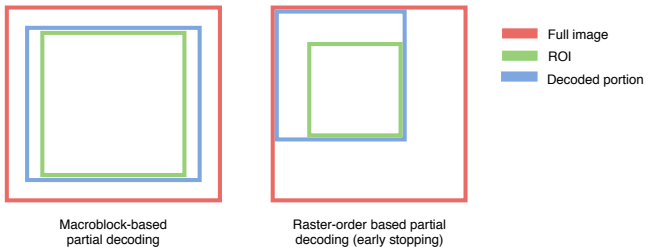

Figure 3: Examples of partial decoding for images. On the left, the ROI is the central crop of the image. For JPEG images, Smol can decode only the macroblocks that intersect the ROI. For image formats that do not allow for independently decoding macroblocks, Smol can partially decode based on raster order (right). Thus, only the blue portion need be decoded. As decoding is generally more expensive than other parts of the preprocessing pipeline, partial decoding can significantly improve throughput.

instantiations of partial decoding in popular visual compression formats and provide a list of popular visual data compression formats and which features they contain in Table 5. We then describe how to use these decoding features for optimized preprocessing.

First, for the JPEG image compression standard, each 8x8 block, or *macroblock*, in the image can be decoded independently (partial decoding) [66]. Second, the H.264 and HEVC video codecs contain deblocking filters, which can be turned off at the decoding stage for reduced computational complexity at the cost of visual fidelity (reduced fidelity decoding) [60, 67]. Third, the JPEG2000 image compression format contains "progressive" images, i.e., downsampled versions of the same image, that can be partially decoded to a specific resolution (multi-resolution decoding) [63].

Smol accepts as an optional input an ROI for a given image. If an ROI is specified, Smol will only decode the parts of the image necessary to process the ROI.

Partial decoding. We present two methods of partially decoding visual data. We show examples of each in Figure 3.

ROI decoding. When only a portion of the image is needed, e.g., for central cropping or when selecting a region of interest (ROI), only the specified portion of the image need be decoded. To decode this portion of the image, 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. This procedure is formalized in Algorithm 1.

Early stopping. For compression formats that do not explicitly allow for partial decoding, Smol can terminate decoding on parts of the image that are not necessary. For example, if only the top $N \times N$ pixels are required for inference, Smol will terminate decoding after decoding the top $N \times N$ pixels.

Reduced-fidelity decoding. Several visual compression formats contain options for reduced fidelity decoding. While there are several ways to reduce the fidelity of decoding for decreased preprocessing costs, we focus on methods that are easily specified with existing decoding APIs. Specifically, we explore reduced fidelity in the form of disabling the deblocking filter. Smol will profile the accuracy of

Algorithm 1 Partial JPEG decoding for a fixed DNN input resolution of 224×224

Ensure: $img \in (3,h,w)$
$w',h' \leftarrow RatioPreservingResize(img)$
$l,t = \frac{w'-224}{2}, \frac{h'-224}{2}$
$r,b = l+224, t+224$
$scale \leftarrow \frac{\min(h,w)}{224}$
$l',r',t',b' = scale \cdot [l,r,t,b]$
img.SetCropline(l', r', t')
while rowIdx $\leq b'$ **do**
  img[rowIdx] = ReadScanlines(rowIdx)
  rowIdx += 1
end while

Format	Type	Low-fidelity features
JPEG	Image	Partial decoding
PNG, WebP	Image	Early stopping
HEIC/HEVC	Image/Video	Reduced fidelity decoding
H.264	Video	Reduced fidelity decoding
VP8	Video	Reduced fidelity decoding
VP9	Video	Reduced fidelity decoding

Table 5: A list of popular visual data formats and their low-fidelity features. Many popular formats contain methods of decoding parts of the visual data, including the popular JPEG, H.264, and HEVC formats.

the specialized and target NNs with and without the deblocking filter and choose the option that maximizes throughput.

7 DISCUSSION ON HARDWARE AND POWER

Overview. Throughout, we use the `g4dn.xlarge` instance as our testing environment. The `g4dn.xlarge` instance has a single NVIDIA T4 GPU, 4 vCPU cores (which are hyperthreads), and 15GB of RAM. The CPU type is the Intel Xeon Platinum 8259CL CPU, which is a proprietary CPU developed specifically for this instance type. Its power draw is 210 watts, or 4.375 watts per vCPU core.

Discussion. We note that there are other `g4dn` instances which contain a single T4 GPU and 8, 16, 32, and 64 vCPU cores. These other instances types could be used to improve the preprocessing throughput by using more cores.

Nonetheless, our speedup numbers can be converted to cost savings when considering other instance types and our conclusions remain unchanged with respect to cost. For example, a 3× improvement in throughput of preprocessing can be translated to using 3× fewer cores. Furthermore, the cost of additional cores dominates: around 3.4 vCPU cores is the same price as the T4 when estimating the cost of vCPU cores and the T4 (see below).

Using these price and power estimates, we estimate the relative price and power of preprocessing and DNN execution. Preprocessing is significantly more expensive than DNN execution for the ResNet-50, both in terms of cost ($0.218 vs $2.37 per hour) and power draw (70W vs 161W), for the configuration in Figure 1. For ResNet-18, these differences are more prominent: $0.218 vs $6.501 and 444W vs 70W for price and power respectively.

GPU	Release date	Throughput (im/s)
K80	2014	159
P100	2016	1,955
T4	2019	4,513
V100	2017	7,151
A100	2020	23,973

Table 6: Throughput of ResNet-50 on GPU accelerators. Throughput has improved by over 94× in three years and will continue to improve. The T4 is an inference optimized accelerator that is significantly more power efficient than the V100, but contains similar hardware units.

Dataset	# of classes	# of train im.	# of test im.
`bike-bird`	2	23k	1k
`animals-10`	10	25.4k	2.8k
`birds-200`	200	6k	5.8k
`imagenet`	1,000	1.2M	50K

Table 7: Summary of dataset statistics for the still image datasets we used in our evaluation. The datasets range in difficulty and number of classes. `bike-bird` is the easiest dataset to classify and `imagenet` is the hardest to classify.

Core price estimation. We estimate the price per vCPU core using a linear interpolation, assuming the T4 is a fixed price and the remaining price is split equally among the cores. Using this method, we find that the hourly cost of the T4 accelerator is approximately \$0.218 and the cost of a single vCPU core is approximately \$0.0639. The R^2 value of this fit is 0.999. Thus, approximately 3.4 vCPU cores is the same hourly price of a T4.

Trends in Hardware Acceleration for DNNs. We benchmarked ResNet-50 throughput on the K80, P100, T4, and V100 GPUs to show the effect of improved accelerators on throughput; we further show the reported throughput of the A100. We used a batch size of 64 for experiments on GPUs. As shown in Table 6, throughput has improved by 44× in three years. Furthermore, accelerators will become more efficient, e.g., with the newly released A100 accelerator.

8 EVALUATION

We evaluated Smol on eight visual datasets and show that Smol can outperform baselines by up to 5.9× for image datasets and 10× for video datasets at a fixed accuracy level.

8.1 Experimental Setup

Overview. We evaluate our optimizations on four image datasets and four video datasets. The task for the image datasets is image classification. The task for the video datasets is an aggregation query for the number of target objects per frame. For classification, we use accuracy and throughput as our primary evaluation metrics. For the aggregation queries, we measure query runtime as the error bounds were respected. We describe Smol's implementation in an extended version of this paper [38].

Datasets. We use `bike-bird` [9], `animals-10` [5], `birds-200` [65], and `imagenet` [19] as our image datasets. These datasets vary in difficulty and number of classes (2 to 1,000). In contrast, several recent systems study only binary filtering [6, 10, 42]. We summarize dataset statistics in Table 7. We used thumbnails encoded in a standard short size of 161 in PNG, JPEG ($q=75$), and JPEG ($q=95$).

For the video datasets, we used `night-street`, `taipei`, `amsterdam`, and `rialto` as evaluated by BlazeIt [34]. We used the original videos as evaluated by BlazeIt and further encoded the videos to 480p for the low-resolution versions.

Model configuration and baselines. For Smol, we use the standard configurations of ResNets (18, 34, and 50). We find that these models span a range of accuracy and speed while only requiring training three models. We note that if further computational resources are available at training time, further models could be explored.

Image datasets. For the image datasets, we use the following two baselines. First, we use standard ResNets and vary their depths, specifically choosing 18, 34, and 50 as these are the standard configurations [28]. We refer to this configuration as the naive baseline; the naive baseline does not have access to other image formats. Second, we use Tahoma as our other baseline, specifically a representative set of 8 models from Tahoma cascaded with ResNet-50, our most accurate model. We choose 8 models due to the computational cost of training these models, which can take up to thousands of GPU hours for the full set of models. We use ROI decoding for Smol as these datasets use central crops.

Video datasets. We used the original BlazeIt code, which uses a "tiny ResNet" as the specialized NN and a state-of-the-art Mask R-CNN [27] and FGFA [72] as target networks. We replicate the exact experimental conditions of BlazeIt, except we use Smol's optimized runtime engine, which is substantially more efficient than BlazeIt.

Hardware environment. We use the AWS `g4dn.xlarge` instance type with a single NVIDIA T4 GPU attached unless otherwise noted. The `g4dn.xlarge` has 4 vCPU cores with 15 GB of RAM. A vCPU is a hyperthread, so 4 vCPUs consists of 2 physical cores. Compute intensive workloads, such as image decoding, will achieve sublinear scaling compared to a single hyperthread.

Importantly, the `g4dn.xlarge` instance is approximately cost balanced between vCPU cores and the accelerator (§7).

`g4dn.xlarge` is optimized for DNN inference. Namely, the T4 GPU is significantly more power efficient than GPUs designed for training, e.g., the V100. However, they achieve lower throughput as a result; our results are more pronounced when using the V100 (e.g., using the `p3.2xlarge` instance). We further describe our choice of hardware environment in §7. We note that our baselines use CPU decoding as a case study, as not all visual formats are supported by hardware decoders, e.g., the popular HEIC (used by all new iPhones) and WebP (used by Google Chrome) formats. Finally, we note that throughputs can be converted to power or cost by using more vCPU cores, but we use a single hardware environment for ease of comparison.

Further experiments. Due to limited space, we include experiments comparing Smol to other frameworks in an extended version of this paper [38].

8.2 Cost Models and Benchmarking Smol

We further investigated the efficiency of pipelining in Smol and our choice of using min in cost modeling (§4). To study these, we measured the throughput of Smol when only preprocessing, only executing the DNN computational graph, and when pipelining both stages.

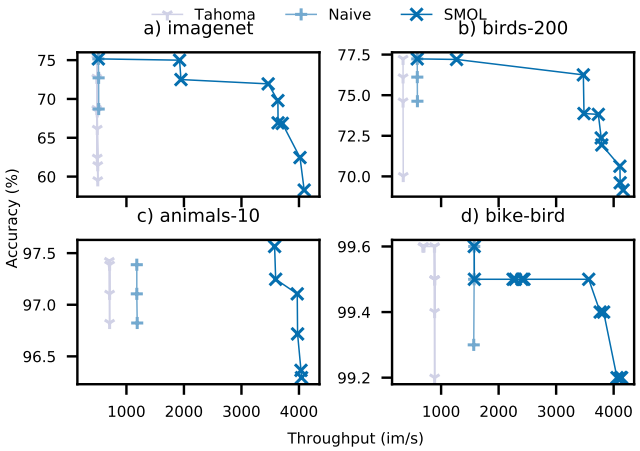

Figure 4: Throughput vs accuracy for the naive baseline, Tahoma, and Smol on the four image datasets (Pareto frontier only). Smol can improve throughput by up to 5.9× with no loss in accuracy. Furthermore, Smol can improve the Pareto frontier compared to both baselines.

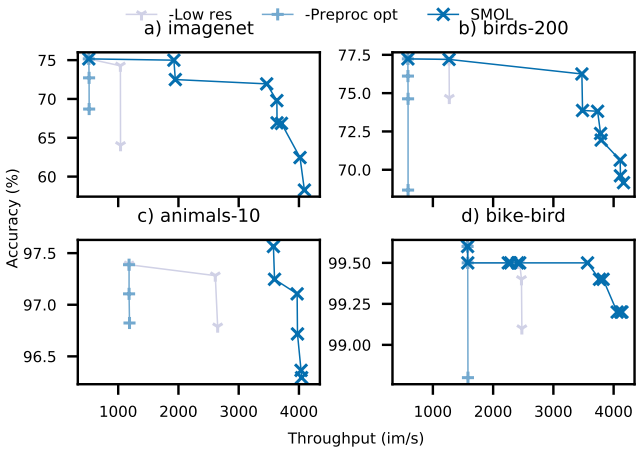

Figure 5: Lesion study for image datasets in which we individually removed the preprocessing optimizations and low-resolution data (Pareto frontier only). As shown, both optimizations improve the Pareto frontier for all datasets.

We first consider low-resolution images (JPEG $q=75$) to ensure the system was under full load. Preprocessing, DNN execution, and end-to-end inference achieve 5.9k, 4.2k, and 3.6k im/s respectively. Even at full load, Smol only incurs a 16% overhead compared to the throughput predicted by its cost model. In contrast, Tahoma's cost model would predict a throughput of 2.5k im/s, a 30% error.

Furthermore, across all ResNet-50 configurations, Smol's cost model (i.e., min) achieves the lowest error compared to other heuristics (i.e., DNN execution only and sum). Its average error is 5.9%, compared to 217% (DNN execution only) and 23% (sum).

8.3 Image Analytics Experiments

End-to-end speedups. We evaluated Smol and baselines (Tahoma and standard ResNets on full resolution data) on the image datasets shown in Table 7.

We first investigated whether Smol outperforms baselines when all optimizations were enabled. We plot in Figure 4 the Pareto frontier of baselines and Smol for different input format and DNN configurations. As shown, Smol can improve throughput by up to 5.9× with no loss in accuracy relative to ResNet-18 and up to 2.2× with no loss in accuracy relative to ResNet-50. Furthermore, Smol can improve the Pareto frontier compared to all baselines. Notably, Tahoma's specialized models performs poorly on complex tasks and are bottlenecked on image preprocessing.

Importantly, we see that the naive baselines (i.e., all ResNet depths) *are bottlenecked by preprocessing* for all datasets. Any further optimizations to the DNN execution alone, including model compression, will *not improve end-to-end throughputs*. The differences in baseline throughputs are due to the native resolution and encoding of the original datasets: `birds-200` contains the largest average size of images. The throughput variation between ResNets depths is due to noise; the variation is within margin of error.

While we show below that both low resolution data and preprocessing optimizations contribute to high throughput, we see that Smol's primary source of speedups depends on the dataset. First, Smol can achieve the same or higher accuracy by simply using low resolution data for some `bike-bird` and `animals-10`. Second, for `imagenet`, a fixed model will result in slightly lower accuracy ($\leq$ 1%) when using lossless image compression. However, when using a *larger* model, Smol can recover accuracy.

Comparison against Tahoma. Tahoma underperforms the naive solution of using a single, accurate DNN for preprocessing bound workloads. This is primarily due to overheads in cascades, namely coalescing and further preprocessing operations. Specifically, Tahoma cascades a small DNN into a larger DNN. These smaller DNNs are less accurate than the larger DNNs and thus require many images to be passed through the cascade for higher accuracy, especially on the more complex tasks. The images that are passed through must be copied again and further resized if the input resolutions are different.

Factor analyses and lesion studies. We further investigated the source of speedups of Smol's optimizations by performing factor analyses and lesion studies.

We performed a lesion study by individually removing the 1) preprocessing DAG optimizations and 2) low-resolution data from Smol. As shown in Figure 5, removing either optimization shifts the Pareto frontier.

We performed a factor analysis by successively adding the preprocessing DAG and low-resolution optimizations to Smol. As shown in Figure 6, both optimizations improve throughput. We further see that the optimizations are task-dependent: the easiest task (`bike-bird`) can achieve high throughput at fixed accuracies with only the preprocessing optimizations. However, many real-world tasks are significantly more complicated than binary classification of birds and bikes.

We also performed a lesion study (Figure 7) and factor analysis (Figure 8) for Smol's systems optimizations. We performed these analyses for ResNet-50 with full resolution and 161 short-side PNG images on the ImageNet dataset to ensure DNN execution was not

Format	Acc (reg train, 50)	Acc (low-resol train, 50)	Acc (reg train, 34)	Acc (low-resol train, 34)
Full resol	75.16%	57.72%	72.72%	64.76%
161, PNG	70.92%	75.00%	68.30%	72.50%
161, JPEG (q=95)	68.93%	71.94%	66.92%	69.79%
161, JPEG (q=75)	64.02%	63.23%	62.45%	62.45%

Table 8: Effect of training procedure and input format on accuracy for ResNet-50 and ResNet-34 on `imagenet`, the most difficult dataset. Smol can achieve an accuracy throughput trade-off by simply changing the input format, e.g., low-resolution ResNet-50 (low-resol train, 50) on 161, JPEG (q=95) achieves approximately the same accuracy as ResNet-34 (reg train, 34) on full resolution data (full resol), namely 71.94% accuracy compared to 72.72% accuracy. Smol can also achieve no loss in accuracy for easier datasets (e.g., `bike-bird`).

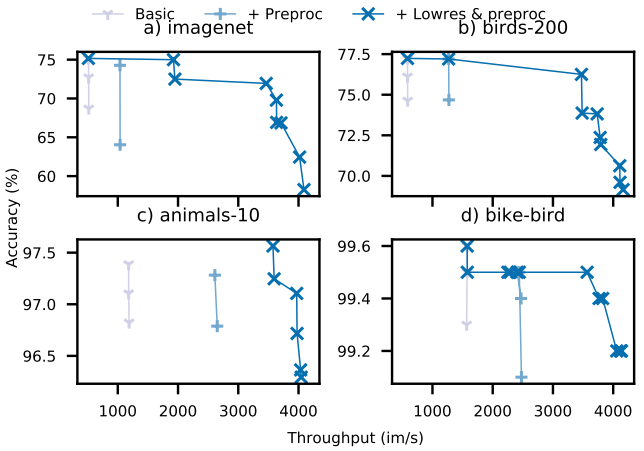

Figure 6: Factor analysis for image datasets in which we successfully add the preprocessing optimizations and then the low-resolution data (Pareto frontier only). Both optimizations improve the Pareto frontier for all datasets.

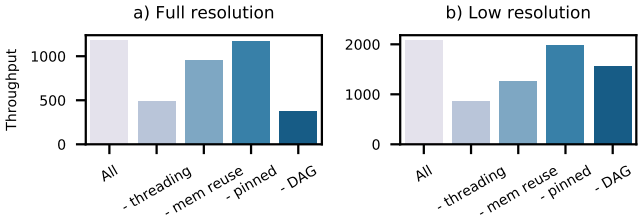

Figure 7: Lesion study of Smol's systems optimizations for full and low resolution images, where optimizations are removed individually. All factors contribute to performance.

the bottleneck. As shown, all systems optimizations improve performance. Furthermore, certain optimizations (e.g., DAG vs threading) contribute more to low resolution and full resolution performance.

Effect of training procedure. We investigated the effect of the training procedure for low-resolution input formats. We trained ResNet-50 on: 1) full resolution, 2) 161 short-side PNG, 3) 161 short-side JPEG (q=95), and 4) 161 short-side JPEG (q=75).

We show the accuracy of these conditions in Table 8 for `imagenet`, our hardest dataset. As shown, low-resolution aware training can nearly recover the accuracy of full resolution data even on this difficult dataset. Low-resolution training can fully recovery accuracy on `bike-bird` and `animals-10`.

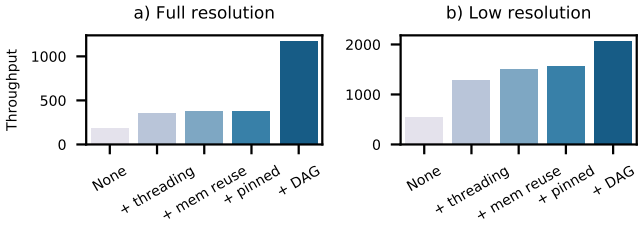

Figure 8: Factor analysis of Smol's systems optimizations for full and low resolution images, where optimizations are added in sequence. All factors contribute to performance.

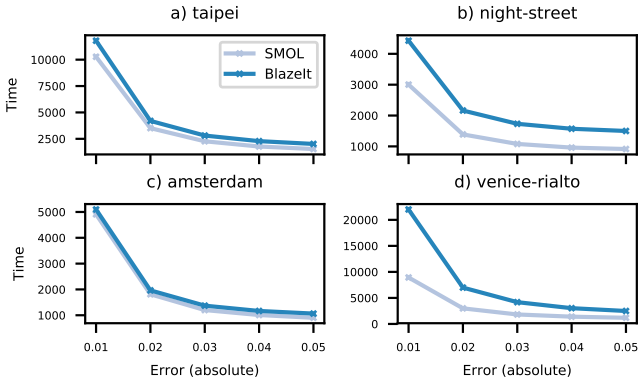

Figure 9: Query execution time vs requested error for BlazeIt and Smol on the four video datasets we evaluated. As shown, Smol consistently outperforms BlazeIt by using more accurate specialized NNs, which reduces sampling variance, and lower resolution data, which reduces preprocessing costs.

8.4 Video Analytics Experiments

We evaluated Smol on the four video datasets described above. We used the exact experimental configuration from BlazeIt as the baseline, with the exception of executing BlazeIt's specialized NNs in Smol's optimized runtime engine. Smol's runtime engine is substantially more efficient that BlazeIt's.

As shown in Figure 9, Smol can improve throughput by up to 2.5× at a fixed error level. Furthermore, Smol outperforms BlazeIt in all settings. Smol's primary speedups for `night-street` and `rialto` come from more accurate, but more expensive specialized NNs. Despite the specialized NNs being more expensive, they reduce sampling variance more, as they are more accurate. As a result,

Condition	vCPUs	Throughput (im/s)	Cost (¢/ 1M images)
Opt	4	1927	7.58
No opt	4	377	38.75
Opt	8	3756	5.56
No opt	8	634	32.92
Opt	16	4548	7.35
No opt	16	1165	28.68

Table 9: Throughput and cost of Smol with and without optimizations at variable number of vCPU cores to achieve 75% accuracy on ImageNet. While increasing the number of cores improves throughput, Smol's optimizations decrease the per-image cost in all cases.

fewer samples are necessary for a fixed error target. Smol's primary speedups for `taipei` and `amsterdam` come from leveraging low resolution video, as it is cheaper to preprocess.

8.5 Cost Analysis

We analyze how preprocessing affects the cost of inference and how preprocessing scales with the number of vCPU cores on public cloud instances. We show the throughput and cost of Smol with and without optimizations to reach an accuracy of 75% on ImageNet in Table 9. We use the prices of the AWS `g4dn` instances. Increasing the number of vCPU cores improves throughput up until matching the throughput of ResNet-50 on the T4. Nonetheless, Smol is the most cost effective by up to 5× per image. We also see that Smol scales nearly linearly with the number of cores, indicating its efficiency.

9 RELATED WORK

Visual analytics systems. Contemporary visual analytics systems leverage DNNs for high accuracy predictions and largely focus on optimizing the cost of executing these DNNs [6, 10, 35, 36, 42]. 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.

Other video analytics systems, such as Scanner [53] or VideoStorm [71] optimize queries as a black box. These systems aim to use all available hardware resources but do not jointly optimize preprocessing and DNN execution.

Systems for optimized DNN execution and serving. Researchers have proposed compilers for optimizing DNN computation graphs, including TensorRT [2] and others [12, 39, 54]. These compilers generally cannot jointly optimize preprocessing and DNN execution. Furthermore, as they generate more efficient code, preprocessing bottlenecks will only increase.

Other optimizations for DNN execution. Researchers have proposed machine learning techniques from model distillation [29] to model compression [26] to reduce the cost of DNN execution. These techniques generally take a given DNN architecture and improve its accuracy or speed. We are unaware of work in the machine learning literature for preprocessing-aware optimizations. These optimizations further improve DNN throughput, but will only increase the gap between preprocessing and DNN execution.

Optimizing DNN preprocessing. To our knowledge, the only system that focuses on optimizing DNN preprocessing is NVIDIA DALI [45]. However, DALI optimizes preprocessing for DNN training and focuses on data augmentation. We show that this results in suboptimal performance in the inference setting. Other contemporaneous work benchmarks end-to-end DNN-based applications [58].

Accelerators for DNNs. Due to the computational cost of DNN computational graphs, researchers and companies have created accelerators for DNN execution. These work exclusively focus on optimizing DNN execution: no paper we surveyed measured *end-to-end* execution time [3, 4, 11, 13–15, 21, 22, 24, 25, 32, 33, 40, 44, 48, 49, 51, 55, 56, 59, 64]. Until very recently, executing the DNN computational graph was the overwhelming bottleneck in DNN execution, but we show evidence that trend has reversed (§2). Thus, we believe it is critical to reason about *end-to-end* performance.

10 DISCUSSION

We have shown that jointly optimizing preprocessing and DNN execution can give large improvements in throughput for analytics on existing visual media compression formats. However, we believe that our techniques can be further extended as future work.

First, similar techniques can likely be applied to other multimedia. For example, many audio compression techniques have similar features to visual compression. Furthermore, audio compression also has a natural trade-off between fidelity and quality.

Second, as DNN accelerators become more efficient, we believe there is promise in the joint design between compression and DNN network design. For example, the massively parallel arithmetic units in DNN accelerators are not well suited to accelerate standard entropy decoding in existing visual formats. However, they may be efficient for other forms of compression. Furthermore, compression algorithms could be designed to improve DNN accuracy.

11 CONCLUSION

In this work, we show that preprocessing can be the bottleneck in end-to-end DNN inference. We show that the preprocessing costs are accounted for incorrectly in cost models for selecting models in visual analytics applications. To address these issues, we build Smol, an optimizing runtime engine for end-to-end DNN inference. Smol contains two novel optimization for end-to-end DNN inference: 1) an improved cost model for estimating DNN throughput and 2) joint optimizations for preprocessing and DNN execution that leverage low-resolution data. We evaluate Smol and these optimizations and show that Smol can achieve up to 5.9× improvements in throughput.

ACKNOWLEDGMENTS

We thank Sahaana Suri, Kexin Rong, and members of the Stanford Infolab for their feedback on early drafts. This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, Infosys, NEC, and VMware—as well as Toyota Research Institute, Northrop Grumman, Amazon Web Services, Cisco, and the NSF under CAREER grant CNS-1651570. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. Toyota Research Institute ("TRI") provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

REFERENCES

[1] 2018. MLPerf. https://mlperf.org/.

[2] 2019. NVIDIA TensorRT. https://developer.nvidia.com/tensorrt

[3] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O'Leary, Roman Genov, and Andreas Moshovos. 2017. Bit-pragmatic deep neural network computing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 382–394.

[4] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 1–13.

[5] Corrado Alessio. 2019. Animals-10. https://www.kaggle.com/alessiocorrado99/animals10

[6] Michael R Anderson, Michael Cafarella, Thomas F Wenisch, and German Ros. 2019. Predicate Optimization for a Visual Analytics Database. *ICDE* (2019).

[7] Elizabeth Arens. 2019. Always Up-to-Date Guide to Social Media Image Sizes. https://sproutsocial.com/insights/social-media-image-sizes-guide/

[8] Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer.

[9] Tom B Brown, Nicholas Carlini, Chiyuan Zhang, Catherine Olsson, Paul Christiano, and Ian Goodfellow. 2018. Unrestricted adversarial examples. *arXiv preprint arXiv:1809.08352* (2018).

[10] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David Andersen, Michael Kaminsky, and Subramanya Dulloor. 2019. Scaling Video Analytics on Constrained Edge Nodes. *SysML* (2019).

[11] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. 2010. A dynamically configurable coprocessor for convolutional neural networks. *ACM SIGARCH Computer Architecture News* 38, 3 (2010), 247–257.

[12] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. {TVM}: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 578–594.

[13] Yunji Chen, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2016. DianNao family: energy-efficient hardware accelerators for machine learning. *Commun. ACM* 59, 11 (2016), 105–112.

[14] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 367–379.

[15] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 27–39.

[16] François Chollet et al. 2015. Keras.

[17] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Chris Re, and Matei Zaharia. 2018. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. *arXiv preprint arXiv:1806.01427* (2018).

[18] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2017. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. *Training* 100, 101 (2017), 102.

[19] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[20] Steven Eliuk, Cameron Upright, Hars Vardhan, Stephen Walsh, and Trevor Gale. 2016. dMath: Distributed Linear Algebra for DL. *arXiv preprint arXiv:1611.07819* (2016).

[21] Clément Farabet, Berin Martini, Benoit Corda, Polina Akselrod, Eugenio Culurciello, and Yann LeCun. 2011. Neuflow: A runtime reconfigurable dataflow processor for vision. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*. IEEE, 109–116.

[22] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. 2018. A configurable cloud-scale DNN processor for real-time AI. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 1–14.

[23] T Gale, S Eliuk, and C Upright. 2017. High-Performance Data Loading and Augmentation for Deep Neural Network Training. In *GPU technology conference 2017.*

[24] Vinayak Gokhale, Jonghoon Jin, Aysegul Dundar, Berin Martini, and Eugenio Culurciello. 2014. A 240 g-ops/s mobile coprocessor for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 682–687.

[25] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 243–254.

[26] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[27] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *ICCV*. IEEE, 2980–2988.

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

[29] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[30] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[31] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. 2018. Focus: Querying Large Video Datasets with Low Latency and Low Cost. *OSDI* (2018).

[32] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 1–12.

[33] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, and Andreas Moshovos. 2016. Stripes: Bit-serial deep neural network computing. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–12.

[34] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. BlazeIt: optimizing declarative aggregation and limit queries for neural network-based video analytics. *Proceedings of the VLDB Endowment* 13, 4 (2019), 533–546.

[35] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. Challenges and Opportunities in DNN-Based Video Analytics: A Demonstration of the BlazeIt Video Query Engine. CIDR.

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

[37] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Approximate Selection with Guarantees using Proxies. *PVLDB* (2020).

[38] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. 2020. Jointly Optimizing Preprocessing and Inference for DNN-based Visual Analytics. *arXiv preprint arXiv:2007.13005* (2020).

[39] Chris Leary and Todd Wang. 2017. XLA: TensorFlow, compiled. *TensorFlow Dev Summit* (2017).

[40] Shuangchen Li, Dimin Niu, Krishna T Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2017. Drisa: A dram-based reconfigurable in-situ accelerator. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 288–301.

[41] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*. Springer, 21–37.

[42] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating Machine Learning Inference with Probabilistic Predicates. In *SIGMOD*. ACM, 1493–1508.

[43] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. 2019. Mlperf training benchmark. *arXiv preprint arXiv:1910.01500* (2019).

[44] Bert Moons and Marian Verhelst. 2016. A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets. In *VLSI Circuits (VLSI-Circuits), 2016 IEEE Symposium on*. IEEE, 1–2.

[45] NVIDIA. 2019. NVIDIA DALI. https://docs.nvidia.com/deeplearning/sdk/dali-developer-guide/docs/index.html

[46] NVIDIA. 2020. NVIDIA T4 Tensor Core GPU for AI Inference. https://www.nvidia.com/en-us/data-center/tesla-t4/

[47] Shoumik Palkar, James J Thomas, Anil Shanbhag, Deepak Narayanan, Holger Pirk, Malte Schwarzkopf, Saman Amarasinghe, Matei Zaharia, and Stanford InfoLab. 2017. Weld: A common runtime for high performance data analytics. In *Conference on Innovative Data Systems Research (CIDR)*.

[48] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, Vol. 45. ACM, 27–40.

[49] Seong-Wook Park, Junyoung Park, Kyeongryeol Bong, Dongjoo Shin, Jinmook Lee, Sungpill Choi, and Hoi-Jun Yoo. 2015. An energy-efficient and scalable deep learning/inference processor with tetra-parallel MIMD architecture for big data applications. *IEEE transactions on biomedical circuits and systems* 9, 6 (2015), 838–848.

[50] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).

[51] Maurice Peemen, Arnaud AA Setio, Bart Mesman, Henk Corporaal, et al. 2013. Memory-centric accelerator design for Convolutional Neural Networks.. In *ICCD*, Vol. 2013. 13–19.

[52] William B Pennebaker and Joan L Mitchell. 1992. *JPEG: Still image data compression standard*. Springer Science & Business Media.

[53] Alex Poms, William Crichton, Pat Hanrahan, and Kayvon Fatahalian. 2018. Scanner: Efficient Video Analysis at Scale (To Appear). (2018).

[54] PyTorch Team. 2018. The road to 1.0: production ready PyTorch. https://pytorch.org/blog/the-road-to-1_0/

[55] Atul Rahman, Jongeun Lee, and Kiyoung Choi. 2016. Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. IEEE, 1393–1398.

[56] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 267–278.

[57] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2019. Mlperf inference benchmark. *arXiv preprint arXiv:1911.02549* (2019).

[58] Daniel Richins, Dharmisha Doshi, Matthew Blackmore, Aswathy Thulaseedharan Nair, Neha Pathapati, Ankit Patel, Brainard Daguman, Daniel Dobrijalowski, Ramesh Illikkal, Kevin Long, et al. 2020. Missing the Forest for the Trees: End-to-End AI Application Performance in Edge Data Centers. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 515–528.

[59] Yongming Shen, Michael Ferdman, and Peter Milder. 2017. Maximizing CNN accelerator efficiency through resource partitioning. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 535–547.

[60] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. 2012. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on circuits and systems for video technology* 22, 12 (2012), 1649–1668.

[61] Mingxing Tan and Quoc V Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946* (2019).

[62] Mingxing Tan, Ruoming Pang, and Quoc V Le. 2019. Efficientdet: Scalable and efficient object detection. *arXiv preprint arXiv:1911.09070* (2019).

[63] David Taubman and Michael Marcellin. 2012. *JPEG2000 image compression fundamentals, standards and practice: image compression fundamentals, standards and practice*. Vol. 642. Springer Science & Business Media.

[64] Swagath Venkataramani, Ashish Ranjan, Subarno Banerjee, Dipankar Das, Sasikanth Avancha, Ashok Jagannathan, Ajaya Durg, Dheemanth Nagaraj, Bharat Kaul, Pradeep Dubey, et al. 2017. Scaledeep: A scalable compute architecture for learning and evaluating deep networks. In *ACM SIGARCH Computer Architecture News*, Vol. 45. ACM, 13–26.

[65] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. 2011. The caltech-ucsd birds-200-2011 dataset. (2011).

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

[67] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. 2003. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology* 13, 7 (2003), 560–576.

[68] Hao Wu. 2019. Low Precision Inference on GPU. https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9659-inference-at-reduced-precision-on-gpus.pdf

[69] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1492–1500.

[70] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. 2019. VStore: A Data Store for Analytics on Large Videos. In *Proceedings of the Fourteenth EuroSys Conference 2019*. ACM, 16.

[71] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *NSDI*, Vol. 9. 1.

[72] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. 2017. Flow-guided feature aggregation for video object detection. *arXiv preprint arXiv:1703.10025* (2017).