Performance and Memory Trade-offs of Deep Learning Object Detection in Fast Streaming High-Definition Images

Aishwarya Srivastava, Dung Nguyen, Siddhant Aggarwal, Andre Luckow, Edward Duffy,
Ken Kennedy, Marcin Ziolkowski, Amy Apon
CLEMSON UNIVERSITY, SOUTH CAROLINA, USA
{aishwas, dungn, skaggar, aluckow, duffy2, kkenned, zziolko, aapon}@clemson.edu

***Abstract*—Deep learning models are associated with various deployment challenges. Inference of such models is typically very compute-intensive and memory-intensive. In this paper, we investigate the performance of deep learning models for a computer vision application used in the automotive manufacturing industry. This application has demanding requirements that are characteristic of Big Data systems, including high volume and high velocity. The application has to process a very large set of high-definition images in real-time with appropriate accuracy requirements using a deep learning-based object detection model. Meeting the run time, accuracy, and resource requirements require a careful consideration of the choice of model, model parameters, hardware, and environmental support. In this paper, we investigate the trade-offs of the most popular deep neural network-based object detection models on four hardware platforms. We report the trade-offs of resource consumption, run time, and accuracy for a realistic real-time application environment.**

I. INTRODUCTION AND MOTIVATING APPLICATION

Deep learning systems have become pervasive in the automotive domain [1]. A key example is visual inspection in automotive manufacturing, which is conducted using camera-based systems and different kinds of classifier, detection, and segmentation methods. The data-intensive nature and computational complexity of these algorithms presents several challenges. Visual inspection requires the deployment of complex algorithms, such as deep learning algorithms, and they may need to be deployed on devices close to the data source and devices to reduce latency and bandwidth restrictions, as well as addressing privacy requirements. Edge and cloud computing platforms are critical in this context for providing the ability to act in real-time and also to support offline processing [2]. Another challenge for our industrial application is the deployment of appropriate tools into the application environment. Finally, model and hardware selection have a significant impact on the runtime and accuracy in data-intensive settings.

The contribution of this paper is to provide a comprehensive study of the trade-offs of runtime performance, memory consumption, classifier accuracy, and hardware selection for object detection in a fast streaming, high definition, image workload application on state-of-the-art hardware platforms. We analyze the trade-offs of popular object detection models on the newest available types of GPU hardware platforms that can be deployed on edge or cloud computing platforms. We focus on designing a robust inference environment that is capable of sustaining high throughput and low latency as well as model accuracy that meets the application requirements. The constraints of our execution environment impose hard deadlines on the completion of the object detection methods. For the purpose of this study, we utilize a set of pre-trained object detectors, and report on test-time performance characteristics: inference time, deployment time, memory footprint, and hardware utilization. Other execution time performance metrics for our application, such as network latency, are studied separately.

The remainder of this paper describes: (i) the test bed and synthetic workload; (ii) the model architectures under consideration and the impacts to accuracy and runtime; (iii) the experimental setup and target performance metrics; and (iv) experimental results and analysis. We include a section on related works and finish with conclusions.

II. SYSTEM TEST BED AND SYNTHETIC WORKLOAD

Complete automated inspection of vehicles based on computer vision imposes requirements on the data collection setup. The system must provide high resolution images that allow inference of a large amount of information and must process these under rigorous time and accuracy constraints. An example processing task is to identify anomalous objects in the images, such as a missing button or a handle that is misaligned. We have designed a test bed and synthetic workload that allow us to study the performance, memory, and accuracy trade-offs for the detection of anomalous objects in the images.

For the visual inspection application, we assume that a single pixel on the image is mapped to 0.1 mm of the inspected area and that the minimal visible portion of the car on the assembly line is approximately 1,300 mm, which can be covered by 13,000 pixels of vertical height. This number of pixels may be achieved by either using a single very high resolution camera or with multiple lower resolution cameras such that the whole height of the car is covered by several images. Our application benefits from the latter solution for

978-1-5386-5035-6/18/$31.00 ©2018 IEEE

several reasons. First, it is more cost effective to use several smaller resolution cameras than to use a single very high resolution camera (at the 13,000 pixel level). Secondly, we have better control over lens distortion at small distance to an object, and a higher frame rate is possible using cameras with a smaller resolution. In addition to these advantages, a multiple camera setup enables much better insight into the depth information for inspection of the geometry.

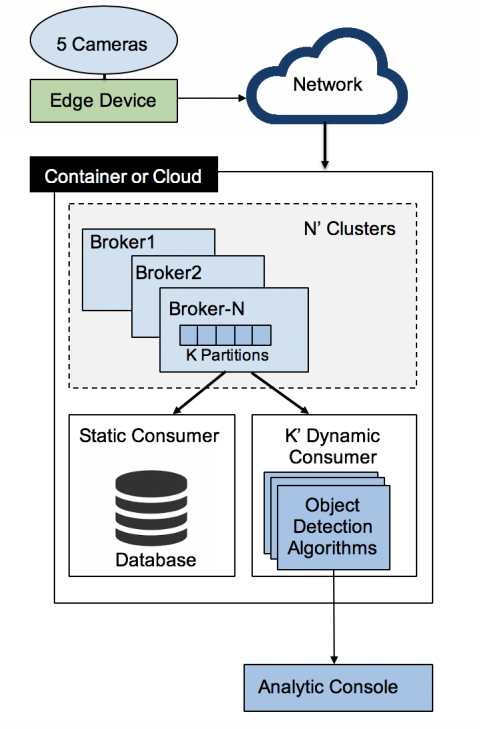


Fig. 1: System Architecture Diagram.

Given these parameters, we have designed a system that includes a set of cameras in a vertical array that together cover the vertical visible portion of the car. Fig. 1 illustrates our system architecture. As a car moves through the assembly line at a fixed rate, camera images are acquired and sent to the image processing infrastructure. A software broker (e.g., Kafka) directs images from the cameras through the network to one or more image processing edge or cloud nodes, each of which can have zero or more GPU processing units. The results of the object detection are available to a human analyst. For real time processing, the results from a car must be available before the next car reaches the inspection point in the assembly line. Images and object detection results are also stored in persistent storage for later analysis.

We have designed a synthetic workload that is representative of the images that would be obtained from real cameras in the car assembly application. Our design assumes that the application includes five cameras with approximately 2,700 pixels of vertical resolution and 2,100 pixels of horizontal resolution each. This resolution is the minimum required for our automotive assembly application. We calculate that, minimally, nineteen camera shots by the five cameras are required for visual inspection of the whole car, for a total of ninety-five images per car.

We process each image by dividing it into tiles with the size matching the smallest native input size of the object detection methods that we consider, which is 300 pixels by 300 pixels. We note that different object detection models, described in the next section, use different native image input sizes. However, to keep the accuracy comparable across the different models, we use the same input size of 300x300 and the same input tiles for all testing of object detection methods. In this paper, we do not include overlap between neighboring tiles, which may affect the accuracy of the detection of small objects.

The synthetic workload is constructed from a collection of images in which every image is composed of tiles of images retrieved from the Common Objects in Context (COCO) data set [3]. Thus, each "camera image" is a single image composed of a set of tiles on a nine by seven grid, providing a consistent number of sixty-three tiles per image. Each tile is 300x300 pixels, creating images that are each 2700x2100 pixels, and ninety-five such images are acquired to provide visual coverage of a whole car. A sample synthetic image used for evaluation and performance testing is shown in Fig 2.

Fig. 2: The left side shows a sample synthetic image. Each image is composed of sixty-three tiles. The complete visual view of a car in the synthetic workload is represented by ninety-five images. The right side is zoomed in on four of the tiles and shows the objects detected.

III. Model Architecture Background

Traditionally, computer vision applications have required complex feature engineering tasks to produce effective feature sets for different kinds of object detection applications. Today, deep learning techniques can be applied directly to raw images without complex feature extraction algorithms. Many common tasks such as object detection can be solved effectively using out-of-the-box deep learning architectures. In our experimental environment, which includes a set of fixed-location cameras, all images are the same size and all models are tested on the same images.

In object detection, a computer vision algorithm tries to detect different objects in an input image and draw a rectangular-shaped *bounding box* that outlines the area of each object. Object detection consists of two different tasks: object classification and bounding box regression. An object with its corresponding box is considered correct if it is both correctly classified and its bounding box exceeds some level of overlapping with manually labelled data, usually at least 50%.

One challenge in object detection is that objects of interest may have different locations within the image and may have different aspect ratios. In a naïve approach, the number of bounding boxes that must be considered is exponentially large with respect to the size of the image. As a result, many different object detection architectures have been proposed that reduce the size of the search space for objects of interest or optimize the search in different ways. These different object detection architectures have different characteristics.

Some models are designed to achieve state-of-the-art performance in accuracy. Several models aim to achieve reasonable accuracy within limited time and computational resources. There are efforts to build deep learning models for special hardware systems, such as FPGAs, or for resource-limited devices such as smartphones. There are multiple degrees of freedom in object detection architectures that affect their accuracy, run time, and resource utilization. The literature provides many details about object detection architectures [4]. Here we list some important degrees of freedom:

1) Meta Architecture: The choice of meta-architecture can have a significant impact on the accuracy and runtime of the model. Meta-architectures in object detection models can be categorized into either single stage or 2-stage models. In single stage models, input images are passed through the networks one time to produce predicted objects and their bounding boxes. In 2-stage models, a first pass of the image produces bounding box proposals and then the box proposals are fed into the second stage to predict objects and recalculate the bounding boxes. Single stage meta-architectures are used for fast, low-latency applications, while 2-stage meta-architectures are used for better accuracy but have longer run times.

We study variants of the single stage model, Single Shot Detector (SSD) [5], including an implementation designed for memory-constrained systems (SSDLite) [6]. We also study variants of several 2-stage models: Region-based Convolutional Neural Network (R-CNN) [7], Faster R-CNN [8], and Region-based Fully Convolutional Networks (R-FCN) [9]. At the time of this paper, Faster R-CNN is considered to be the state-of-the-art meta-architecture in terms of accuracy in object detection, but R-FCN produces comparable accuracy in several common datasets [4].

2) Feature Extractor: At least six feature extractors are reported in [4]. Feature extractors are usually image classification networks that are pre-trained on some common dataset first, and then are used to initialize the complete networks. In our experiments, we study models with a few common feature extractors, including MobileNet [10], Resnet [11], Inception [12], Inception-Resnet [13], and NAS [14]. MobileNet is an efficient deep neural network that is able to produce relatively light-weight networks while maintaining reasonable performance [10]. Deep Residual Network (ResNet), Inception, and Neural Architecture Search network (NAS) all utilize many layers or scales or combinations of scales to achieve higher accuracy. The most accurate model we study uses the NAS feature extractor with the Faster R-CNN meta-architecture (Faster_R-CNN_NAS).

3) Other Features: A few other features often considered in the selection of the model or its parameters: a) Feature layers: These select which layer(s) of the feature extractor to be used in the meta-architecture. Several papers select the top-most, or deepest, convolutional layer, but other papers aggregate multiple layers, even fully-connected layers; b) Box proposals: Different original papers report a different number of proposals, as well as how to select proposals, for example, values from 10 to 300 for an image of 300x300 pixels may be considered; c) Trained image size: Two models we consider, R-FCN and Faster R-CNN are fixed at the shorter edge. SSD is fixed at both edges; d) Stride in the feature extractor: Reports show that stride is an important factor in the trade-off between performance and running time; and, e) Data augmentation: These are methods to transform images, such as rotation or cropping, to make the models more robust.

The original papers typically report a single combination of these options, with variants such as different image resolutions, the numbers and positions of the candidate boxes, the layers from which features are extracted and the number of layers. In this evaluation, we focus on popular methods with different combinations of meta-architectures and feature extractors that have all been pre-trained on the COCO dataset.

IV. Experimental Environment

A. Deep Learning Framework and Object Detection Models

A number of deep learning frameworks appear in the literature and are available for testing, including DeepX [15], PyTorch [16], MXNet [17], and TensorFlow [18]. Of these, we found at the time of our study that only TensorFlow is complete and robust enough to support the range of object detection models that we wish to evaluate.

TensorFlow is an open source machine learning framework with a large user community that includes the support from about fifty companies. Models in TensorFlow include official models that are maintained and tested, and utilize the latest stable TensorFlow API. The TensorFlow model repository also includes research models that are contributed and maintained by individual researchers. New models are continually being added to the repository. We selected the twenty-four research object detection models for use in TensorFlow that were available at the time of the start of this study [18].

B. Hardware Devices

Although a number of low-cost technologies have been promoted for use in edge computing, such as Raspberry PI and mobile phones, our initial testing indicated that the very low

cost platforms were not robust enough to support processing of the high-definition images for our application. We test four models of GPU processors for our application: Nvidia P100, Nvidia V100 PCIe, Nvidia V100 SXM2, and Nvidia Jetson TX2. We also provide results for CPU execution only.

1) Nvidia P100: Nvidia's Tesla P100 (Pascal) architecture, introduced in 2016, contains 3,584 CUDA cores. The card for our tests uses a PCIe bus, has 12GB of RAM, memory bandwidth of 732 GB/s, and a GPU maximum clock rate of 1.33 GHz.

2) Nvidia V100 PCIe: The Tesla V100 (Volta), introduced in 2017, has 5,120 CUDA cores. The card is our tests has 16 GB RAM, memory bandwidth of 900 GB/s, and a GPU maximum clock rate of 1.38 GHz.

In addition to the increased number of CUDA cores, an advantage of the V100 over the P100 is the addition of 640 Tensor cores. A Tensor core uses a fused multiply add (FMA) operation in which two half precision 4x4 matrices are multiplied together, and a half or single precision matrix is added to the result. An FMA operation can be performed within one GPU clock cycle. Some object detection models natively utilize reduced precision in some layers of the algorithms, which can improve execution time without affecting accuracy.

3) Nvidia V100 SXM2: Though similar to PCIe in architecture and number of cores (5,120), Nvidia's Tesla V100 SXM2 uses NVLink as the system interface. The card in our tests has 16 GB memory, memory bandwith of 900 GB/s, and a GPU maximum clock rate of 1.53 GHz. It also has an interconnect bandwidth of 300 GB/s, compared to the PCIe counterpart which offers 32 GB/s.

4) Nvidia Jetson TX2: The Nvidia Jetson TX2 has a Pascal GPU with 256 CUDA cores. Memory is shared with main memory and is 8 GB with a bandwidth of 59.7 GB/s. The TX2, along with prior edge devices TK1 and TX1, and the latest edge device Xavier, are designed to run pre-trained models. As such, this paper focuses on evaluating the less-resource demanding MobileNet models on the TX2. We use TX2 in Max-N power mode, where both dual-core Denver processor and a quad-core ARM Cortex-A57 run at maximum clock speed along with the GPU clock speed of 1.30 Ghz.

5) CPU-only execution: CPU-only execution only tests were performed on a compute node with Intel Xeon Gold 6148 CPU at 2.40 GHz without the use of any GPU. Each test runs on all forty cores of a single compute node.

C. Performance Metrics

There are many factors that can affect performance measurements, such as the executions of other processes, the shared utilization of memory bandwidth, or environmental tasks. We set up a clean and isolated environment with no processes that consume system resources other than required system processes.

1) Inference Time: Inference time includes splitting the test image into multiple tiles and making predictions from all sixty-three tiles. A single blank tile is included to provide for sixty-four tiles, so that the largest mini batch size is a power of two. The mini batch size is a parameter to the models that specifies the number of tiles loaded and processed at the same time. Larger batch sizes can enable more efficient use of the GPU memory and cores.

Inference time does not include loading images from persistent storage devices, decoding images, or transforming images into the data format required by the inference engines. Inference time ends when the results of object detection are calculated. We report in this paper the average inference time over one or more runs of processing of ninety-five images while utilizing a clean test environment.

2) GPU Memory Consumption: In the default configuration, Tensorflow consumes all available GPU memory, meaning that the memory utilization is nearly 100% during all run times. Therefore, we examine the memory consumption with TensorFlow configuration `allow_growth=True` in order for the framework to start with the minimum required memory and to allocate more memory when necessary. We define the memory utilization as the amount of GPU memory allocated to evaluate each process. Note that this definition is different from Nvidia's definition, which reports the percentage of maximum memory bandwidth that is currently utilized at each sampling. The total memory allocated on GPU by active context is measured using 'nvidia-smi'. The memory consumption is reported for each model as the maximum amount of allocated memory during each experiment. We sample the allocated memory values with an interval of 0.01 second.

D. Model Accuracy

The common metric to measure accuracy in the computer vision community is mAP (Mean Average Precision). The mAP is a measure of the ratio of correctly defected objects over the total number of objects detected among all images. A higher mAP means that the model has identified more objects correctly.

Most models have reported mAP accuracy in the TensorFlow site. We additionally validated our test environment to confirm the accuracy of the model output on models with published mAP values. To evaluate our results, we use the COCO metrics from the official COCO Python API [19]. These calculate the average precision over multiple Intersection Over Union (IOU) values ranging from 0.50 to 0.95 with a stride of 0.05.

Note that the reported mAP results were calculated on the COCO test data set, but the labels and annotations for the COCO test data set are not available to the public. We performed accuracy tests using the COCO validation data set. Since we have calculated mAP values using a different data set, the values are different, but the results show that the relative accuracy of the models is the same with one exception. A subset of the models was also hand inspected for accuracy. The list of models, sorted by accuracy, is shown in Table I. The model names shown in Table I give the meta-architectures and feature extractors as previously described. Models are grouped into five groups by mAP, as shown in Table I, to facilitate comparative analysis.

TABLE I: Models sorted by reported mAP with tested mAP and grouped by mAP

Model Name	mAP Reported	mAP Tested	mAP Group
faster_rcnn_nas	43	57.7	Highest
faster_rcnn_incept_resnet_v2_atrous	37	45.2	
mask_rcnn_incept_resnet_v2_atrous	36	45.6	High
ssd_resnet50_v1_fpn	35	–	
mask_rcnn_resnet101_atrous	33	41.5	
ssd_mobilenet_v1_fpn	32	–	
faster_rcnn_resnet101	32	40.3	Medium
rfcn_resnet101	30	37.3	
faster_rcnn_resnet50	30	34.5	
mask_rcnn_resnet50_atrous	29	35.7	
faster_rcnn_incept_v2	28	31.1	
mask_rcnn_incept_v2	25	32.7	Low
ssd_inception_v2	24	30.9	
ssdlite_mobilenet_v2	22	27.4	
ssd_mobilenet_v2	22	–	
ssd_mobilenet_v1	21	26.3	Lowest
ssd_mobilenet_v1_ppn	20	–	
ssd_mobilenet_v1_0.75_depth	18	21.7	

The models with mAP not reported on the TensorFlow site are the "low proposal" models. These are expected to have an mAP that is comparable to the corresponding non-low proposal method. However, since we have not confirmed these accuracies the models are not included in Table I. In four cases, the reported mAP is from testing on a newer version of the TensorFlow framework than our test bed and is not comparable to our testing.

V. Experimental Results and Analysis

In this section, we discuss the experimental results. We tested the full range of models and hardware choices. In a few cases, not all results are shown on the charts for space reasons. The results shown are representative of the range of results and are the most likely combinations of models and architecture to be selected for our application. Results for hardware platforms P100, V100 PCIe, V100 SXM2, CPU-only, and TX2 are shown.

A. Inference Time as a Function of Mini Batch Size

We measure the inference time as a function of the TensorFlow mini batch size for each platform. Mini batch size varies as a choice of 1, 2, 4, 8, 16, 32, and 64. The batch size of 64 loads all tiles in the image as a single batch. The size of a mini batch indicates the size of the fourth dimension of the input tensor supplied to the model's computational graphs (the three other dimensions are height, width, and color channels). Larger batch sizes require larger memory to store input tensors, intermediate representations, and output tensors during computations. However, larger batch sizes reduce the amount of communication between operations, thereby reducing inference time. The inference time is reported in seconds and is shown in log scale on most charts. Some higher accuracy models have an out-of-memory error with larger batch sizes and no result is shown in this case on the chart.

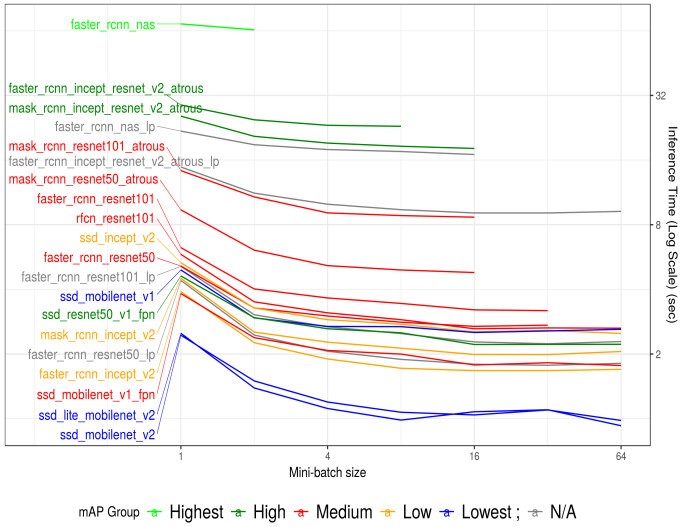


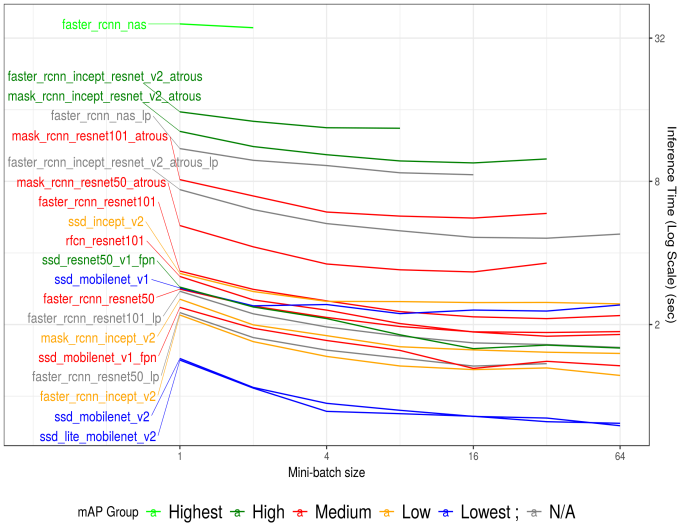
Fig. 3: Inference time as a function of batch size on P100

Fig. 3 shows the inference time as a function of the mini batch size for selected models on the P100 hardware. In general, runtime decreases with larger batch sizes to about a mini batch size of 8, where it tends to level off.

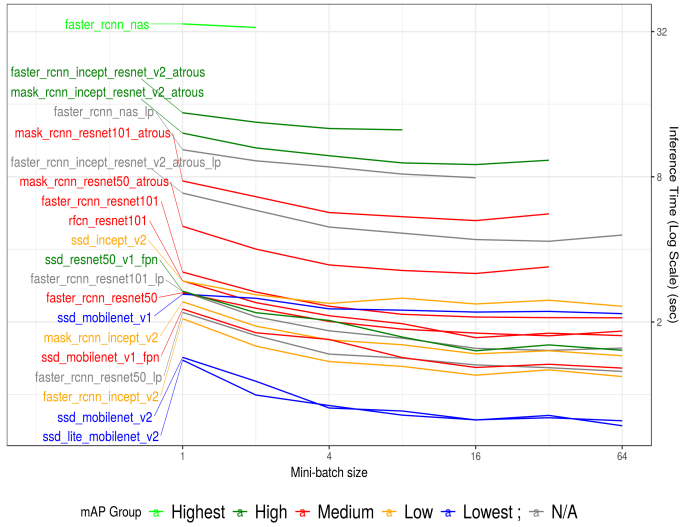
Note that Faster_RCNN_NAS runs out of memory with a batch size of 4 and above on the P100 and on both V100 GPUs, but it is the most accurate model we tested. Some other models with High or Medium accuracy also run out of memory at higher batch sizes. The more accurate models build deeper neural networks that take more memory so fewer batches can fit into memory at a time.

The four "low proposal" models are also shown in Fig. 3, but the accuracy of these is Not Available. Note that these low proposal models have much faster run times than their non-low proposal counterparts. Measuring the accuracy of these models for our application is an area of future inquiry.

Fig. 4a and Fig. 4b show the inference time as a function of the mini batch sizes for the V100 PCIe and V100 SXM2 platforms, respectively. Some similarities and differences can be noted between execution on the P100 and execution on the V100 platforms. The out-of-memory errors for some higher accuracy models occurs at the same batch sizes tested on all three platforms. However, the run times for all models are faster on the V100 platforms than on the P100, and are somewhat faster on the V100 SXM2 than on the V100 PCIe platform. These results are expected since the faster memory bandwidth and clock rate and addition of tensor cores gives the V100 platforms a significant advantage. The fastest models we test are the four MobileNet models. The fastest run time we measure, of all models and hardware choices, is SSD_MobileNet_V2 with V100 SXM2 and a mini batch size of 64. This model has a mean run time of 0.743 seconds to process a single image.



(a) Inference time as a function of batch size on V100 PCIe



(b) Inference time as a function of batch size on V100 SXM2

Fig. 4

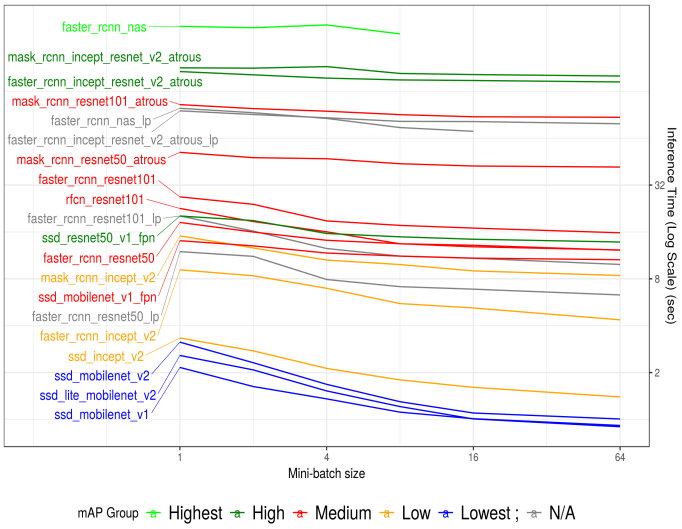


Fig. 5: Inference time as a function of batch size using CPU only

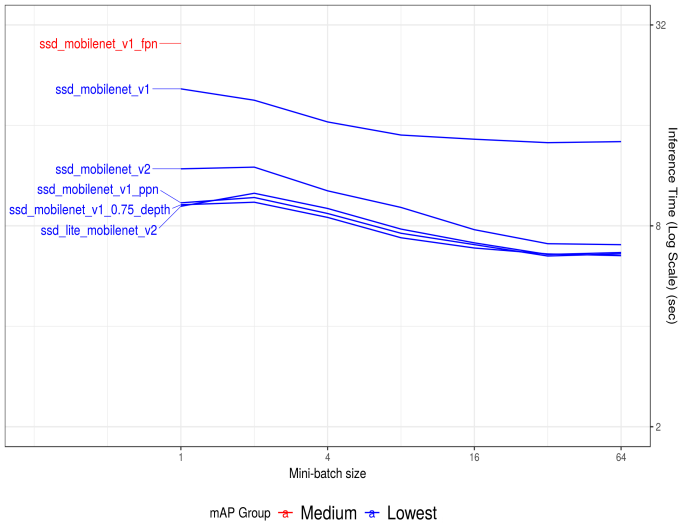
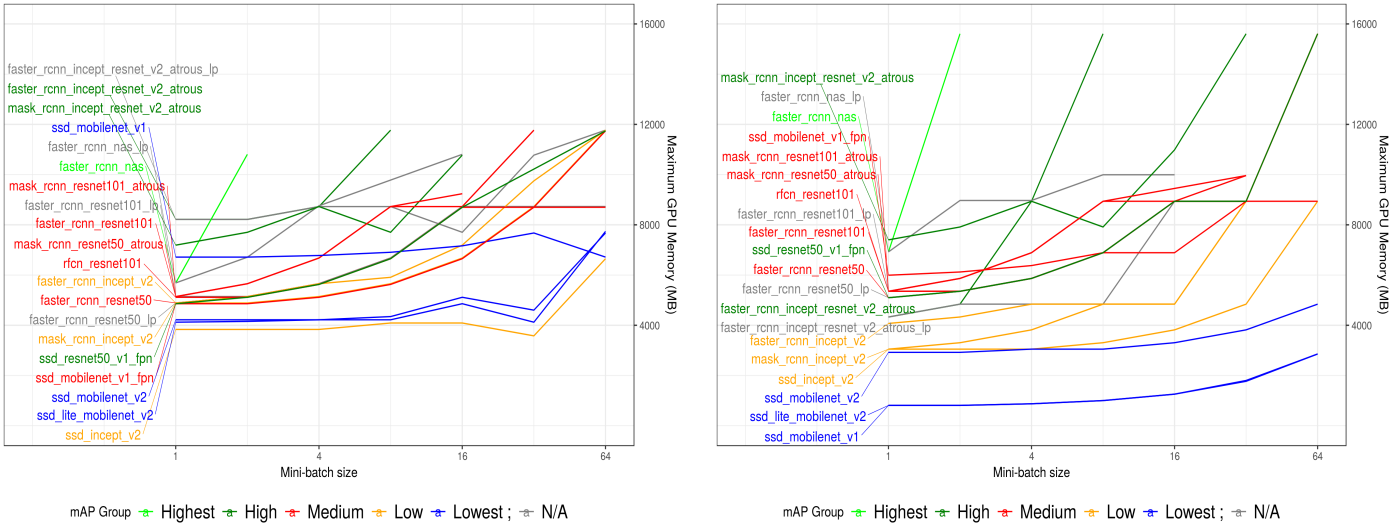


Fig. 6: Inference time as a function of batch size using TX2

Figure 5 shows the inference time as a function of the mini batch sizes for CPU-only execution. The relative ranking of models by run time is similar to the rankings by run time on the GPU platforms. However, note the change of scale on the y-axis. The run times for CPU only are in general much higher for all models than on the GPU platforms. For example, the best run time of Faster_RCNN_NAS using V100 SXM2 is around 32 seconds as compared to the run time on the CPU-only of around 256 seconds, a factor of 8 times slower. For real-time industrial applications, such as ours, selection of hardware includes the ability of its performance to meet timing requirements and, secondly, if the costs of using multiple hardware platforms in parallel to meet all workload demands justify the use of the cheaper, slower platform. We study the costs comparisons separately.

Fig. 6 shows the inference time as a function of the mini batch sizes for TX2 execution. Most models do not execute within our application run time constraints on the TX2, and we only show results for the MobileNet models. The SSD_Mobilenet_v1_FPN model runs out of memory for batch sizes greater than one. The TX2 is designed to be inexpensive, having less memory and other resources restrictions. The execution is much slower for the tested models than on the P100 and V100 platforms, though the platform is less expensive and is useful for many application use cases.



(a) Maximum GPU memory usage as a function of batch size on P100

(b) Maximum GPU memory usage as a function of batch size on V100 PCIe

Fig. 7

B. Memory Consumption as a Function of Mini Batch Size

We measure the GPU memory consumption as a function of the TensorFlow mini batch size using the same parameters as for measuring inference time. Memory consumption is reported in MB for the GPU platforms. As before, higher accuracy models have an out-of-memory error with larger batch sizes and no result is shown in this case on the chart.

Fig. 7a shows the memory consumption as a function of the mini batch size for selected models on the P100 hardware. Fig. 7b shows the memory consumption as a function of the mini batch size for selected models on the V100 PCIe hardware. Memory consumption on the V100 SXM2 hardware is the same as memory consumption on the V100 PCIe hardware and is not shown for space reasons.

On both the P100 and V100 PCIe platforms the SSD models have the smallest memory footprint and all batch sizes fit into memory. Note that the SSD_Mobilenet models, which are shown in blue on the charts since they have the lowest accuracy, use less memory on the V100 PCIe than they do on the P100. On both platforms the SSD_Incept_V2 model uses one of the lower amounts of memory and also has one of the lowest run times along with a Medium level of accuracy. Most Faster_RCNN models run out of memory on the V100 PCIe platform at a batch size of 32 or smaller.

In general, on both platforms, larger batch sizes require more memory but the growth of the memory requirement is not linear. Note that GPU memory on the P100 is 12GB but the V100 PCIe has 16GB. More models run out of memory at lower batch sizes on the P100 than on the V100 PCIe.

We find that the measurements on the P100 are in general less stable than measurements on the V100 PCIe and the V100 SXM2. In this study we have not applied any optmizations to the memory accesses or to the execution by threads within the same warp beyond what is provided by the model codes "out-of-the-box". However, all models either use memory to capacity for both the P100 and the V100 platforms , or show an increase in memory usage for a batch size of 64 over the smaller batch sizes. Applying optimizations of memory usage for selected models is an item of future study.

We do not show results for memory usage for the TX2 since the GPU does not have its own memory on the TX2. It shares the system RAM and is wired to memory controller and generally consumes most of the system memory. We do not break out those numbers in our charts.

C. Inference Time and Memory for Different GPU Platforms

In this section, we study the trade-offs of memory usage and interference time for three different GPU platforms. For this part of the study with each model, we select the batch size that provides the fastest execution time and report the memory usage for that model. Fig. 8 graphs the inference time and memory usage for the models executed on the P100, V100 PCIe, and V100 SXM2 platforms. The reported values are labeled with an ID for each model. Fig. 9 lists the models along with the ID that is used in Fig. 8. The values list the best inference time over all the tested batch sizes and its relative memory usage for each model over every hardware.

The figures show that in general, bigger models with larger inference times achieve better accuracies on both systems. Faster_RCNN_NAS is the model that both achieves highest accuracy and has the longest run time. Also, because of its complex structure, we can only fit a maximum mini batch size of two tiles into the V100 memory.

The fastest models are based on the SSD meta-architecture, but these models have a lower accuracy of 50% of the best model in the best case. With that loss in accuracy, these models

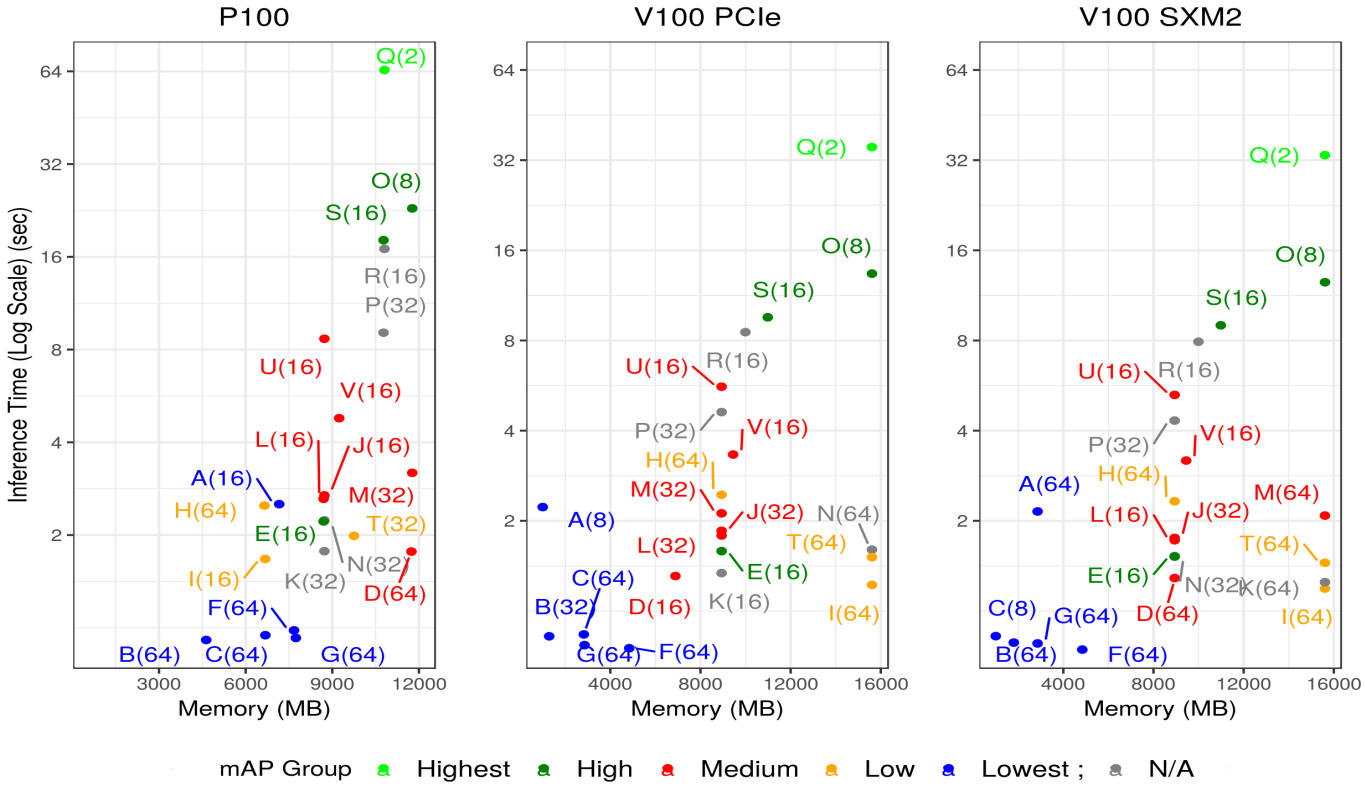


Fig. 8: Best inference time and max memory consumption of models on three GPU devices. The labels used in this figure are shown in Fig. 9. For each point shown the mini batch size that gives the best run time for that model is shown in parentheses. For example, C(64) shown in the lower left of the left figure is the ssd_mobilenet_v1_ppn model run with a mini batch size of 64, which is the fastest for that model on the P100. The left figure shows results for P100, the middle figure shows results for V100 PCIe, and the right figure shows results for V100 SXM2.

ID	Model Name
A	ssd_mobilenet_v1
B	ssd_mobilenet_v1_0.75_depth
C	ssd_mobilenet_v1_ppn
D	ssd_mobilenet_v1_fpn
E	ssd_resnet_50_fpn
F	ssd_mobilenet_v2
G	ssdlite_mobilenet_v2
H	ssd_inception_v2
I	faster_rcnn_inception_v2
J	faster_rcnn_resnet50
K	faster_rcnn_resnet50_lowprop
L	rfcn_resnet101
M	faster_rcnn_resnet101
N	faster_rcnn_resnet101_lowprop
O	faster_rcnn_incept_resnet_v2_atrous
P	faster_rcnn_incept_resnet_v2_atrs_low
Q	faster_rcnn_nas
R	faster_rcnn_nas_lowprop
S	mask_rcnn_incept_resnet_v2_atrous
T	mask_rcnn_incept_v2
U	mask_rcnn_resnet101_atrous
V	mask_rcnn_resnet50_atrous

Fig. 9: List of models and IDs used in Fig. 8.

can process each test image 30 times faster than the slowest models in this study.

Most SSD based models take less than 4 seconds to process a test image on average. SSD_Mobilenet_v1 is the model that achieves smallest memory consumption, even with all tiles processed as a single mini batch. This model is extremely memory efficient, though it is about 3 times slower than the fastest model.

Faster_RCNN models, in general, achieve better accuracy than SSD based models, but with longer run times. Many of these achieve their best inference time with a mini batch size of 16. The most accurate Faster_RCNN model takes nearly 60 seconds to process an image, while the fastest one takes only 2 seconds.

For some models, running on the V100 PCIe is not only faster than the P100 but also requires less memory. Some SSD Lite models can run with less than 4 GB of memory on V100 GPUs, while the same models achieve highest performances at nearly 8 GB on P100. These values are very important in a scenario that an application needs to load multiple models into GPU in the same time. If a model can properly work with less than 4 GB on V100, then it is possible to load four instances of that model to process data in parallel. The P100 has only

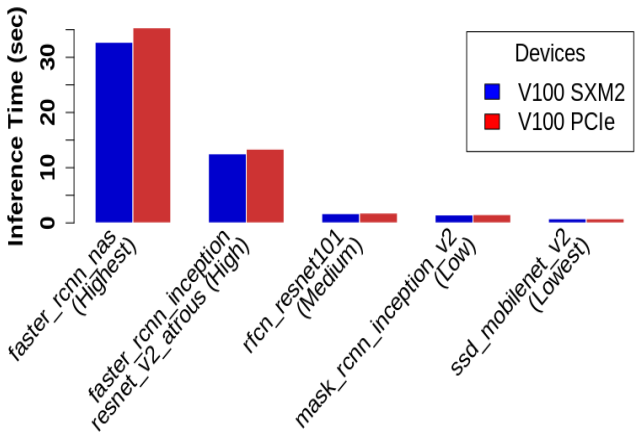


Fig. 10: Inference times on V100 PCIe and V100 SXM2 platforms for representative models from each of the five mAP groups.

has 12 GB of RAM, therefore, we can at most load only one instance of the most lightweight models at a time.

When comparing the two variants of V100 GPUs, PCIe and SXM2, it can be seen that the SXM2 variant produces faster results in all tested models. In fact, having higher clock speeds and faster internal connections between CPU and GPU is an advantage the SXM2 variant. Note that the differences in processing times between the V100 PCIe and V100 SXM2 are relatively small in comparison to the differences between the P100 and V100 GPUs.

D. Discussion of Real-Time Application Constraints

One important aspect of our application is to select a model with the best accuracy possible within the application run time requirements. Fig. 10 can help in the selection of possible models. Fig. 10 shows the run times on the two V100 platforms for selected models from each of the five mAP groups, from highest mAP to the lowest mAP. Fig. 10 shows that the run times are significantly smaller for models with less accuracy. Conversely, if higher accuracy is required by the application then higher run times for model inference are required. For example, with a run time requirement of 1 second, the best accuracy we can achieve for the calculation of an individual model is in the "Lowest" range. However, with a run time requirement of 4 seconds, the best accuracy we can achieve for the calculation of an individual model is a higher value in the "Medium" range. A run time of about 32 seconds for the calculation of an individual model is required to obtain the "Highest" accuracy.

VI. Related Works

MLPerf [20] is a benchmark suite consisting of seven benchmarks including two benchmarks for object detection models. The focus is however on the training aspects of deep learning models and does not cover inference. Dawnbench [21] is a predecessor of MLPerf covering both training and inference. In contrast to older deep learning benchmarks it uses the time-to-accuracy as its primary metric and does not solely focus on accuracy. DeepBench [22] focuses on lower level aspects and hardware, in particular linear algebra level operations required for doing a forward pass on a neural network. As the inference performance also depends on these operations, there is some relevance, but other overheads, such as I/O, framework performance, are not covered by DeepBench. In contrast to the approaches referenced above, this work focuses on specific image inference workloads of a real world application. Our analysis covers further important aspects of the inference pipeline, such as framework overheads, model sizes, and memory footprints.

Huang et al. [4] investigates the inference performance of different object detectors. The authors identify three meta-architectures with several feature extractors and evaluate the trade-off between accuracy and inference time. In contrast to this work, it solely focuses on a generic application use case and does not cover other aspects important for production deployment, such as the memory consumption of the models.

VII. Conclusions and Future Work

Edge inference is a critical component of every deep learning systems enabling us to process large amounts of data with low latencies while preserving privacy. The deployment of deep learning algorithms in production requires the careful understanding of various trade-offs in particular related to the computation and memory requirements of the models and their provided accuracies.

In this paper, we investigate the trade-offs to guide the design of computer vision system for automated inspection. Our results provide to us a selection of appropriate edge hardware. Not surprisingly, models designed for embedded inference, such as MobileNet, are particularly well-suited for edge deployment. However, the ability to deploy also server-scale GPUs on the edge enables us to also utilize high-quality models.

We expect that the requirements for edge inference will significantly increase in the near future as tools will increasingly run complex ensembles of models concurrently (e.g. the detection of a multitude types of issues simultaneously). The provided characteristics will be instrumental for designing and sizing the inference workload as the system evolves.

In the future, we plan to augment edge with cloud resources using scalable streaming services [23]. This will enable us to exploit a higher degree of parallelism by utilizing multiple nodes and GPUs. For this purpose, it is necessary to carefully evaluate the bandwidth and latency needs of the applications against the capabilities of the infrastructure.

The usage of cloud-based resources enables the support of active learning capabilities and the ability to train models that utilize multiple edge cameras as input. We envision various lines of investigation for integrating edge/cloud-based systems: (i) on a infrastructure-level edge resource must be

integrated with cloud resources via streaming. The utilization of specialized cloud hardware, such as Google's TPUs [24] or Microsoft's FPGA [25] requires special consideration both during deployment and runtime; (ii) Multiple model versions potentially trained on different datasets deployed on the edge and/or the cloud to magnify the performance trade-offs described in this paper; (iii) To support active learning approaches, such as federated learning [26], to provide the means to utilize decentralized training resources on the edge, and to combine the results into a global model.

ACKNOWLEDGEMENTS

This research has been supported by National Science Foundation grants #1405767 and #1725573.

REFERENCES

[1] A. Luckow, M. Cook, N. Ashcraft, E. Weill, E. Djerekarov, and B. Vorster. Deep learning in the automotive industry: Applications and tools. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 3759–3768, Dec 2016.

[2] Kevin Scott. The next wave of computing is the intelligent edge and intelligent cloud. https://blogs.microsoft.com/blog/2018/07/23/the-next-wave-of-computing-is-the-intelligent-edge-and-intelligent-cloud/, 2018.

[3] Common Objects in COntext dataset. *http://cocodataset.org*.

[4] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, 2017.

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

[6] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2016.

[8] Ross Girshick. Fast r-cnn. *arXiv preprint arXiv:1504.08083*, 2015.

[9] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[15] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*, page 23. IEEE Press, 2016.

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

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[13] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[14] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2(6), 2017.

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

[17] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[18] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[19] T. Y. Lin and P. Dollar. MS COCO API. https://github.com/cocodataset/cocoapi, 2016.

[20] Mlperf: A broad ml benchmark suite for measuring performance of ml software frameworks, ml hardware accelerators, and ml cloud platforms. https://mlperf.org/, 2018.

[21] C. Coleman, D. Kang, D. Narayanan, L. Nardi, T. Zhao, J. Zhang, P. Bailis, K. Olukotun, C. Re, and M. Zaharia. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. *ArXiv e-prints*, June 2018.

[22] Baidu Research. Deepbench. https://github.com/baidu-research/DeepBench, 2018.

[23] D. Nguyen, A. Luckow, E. Duffy, K. Kennedy, and A. Apon. Evaluation of highly available cloud streaming systems for performance and price. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 360–363, May 2018.

[24] Google. Cloud tpu: Train and run machine learning models faster than ever before. https://cloud.google.com/tpu/, 2018.

[25] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Mahdi Ghandi, Daniel Lo, Steve Reinhardt, Shlomi Alkalay, Hari Angepat, Derek Chiou, Alessandro Forin, Doug Burger, Lisa Woods, Gabriel Weisz, Michael Haselman, and Dan Zhang. Serving dnns in real time at datacenter scale with project brainwave. IEEE, March 2018.

[26] Jakub Konecný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR*, abs/1610.05492, 2016.