MODI: Mobile Deep Inference Made Efficient by Edge Computing

Samuel S. Ogden Tian Guo
Worcester Polytechnic Institute

Abstract

In this paper, we propose a novel mobile deep inference platform, *MODI*, that delivers good inference performance. *MODI* improves deep learning powered mobile applications performance with optimizations in three complementary aspects. First, *MODI* provides a number of models and dynamically selects the best one during runtime. Second, *MODI* extends the set of models each mobile application can use by storing high quality models at the edge servers. Third, *MODI* manages a centralized model repository and periodically updates models at edge locations, ensuring up-to-date models for mobile applications without incurring high network latency. Our evaluation demonstrates the feasibility of trading off inference accuracy for improved inference speed, as well as the acceptable performance of edge-based inference.

1 Introduction

Resource intensive deep learning models are increasingly used in mobile applications [27, 28] to bring features such as object tracking and real-time language translation to end users. However, the use of deep inference with high accuracy is constrained by limits on mobile computation and storage and is exacerbated by the often interactive nature of the workload. To enable mobile-based deep learning, prior work that focused on optimizing models [18, 19, 25, 30], frameworks [7, 12] and hardware [11, 21] has successfully push deep learning into a wide spectrum of devices including IoT devices. However, much of this prior optimization involves *statically* trading off inference accuracy for improved inference time and therefore might not be suitable for dynamic application scenarios.

The goal of balancing inference accuracy and speed requires taking into account multiple factors that are unique to mobile devices. Mobile deep learning has dynamic inference requirements including varying input data, battery limitations and unpredictable network conditions. However, current mobile deep inference is often executed using static configurations, such as a single on-device model or a remote server. These static methods will often lead to sub-optimal performance when faced with dynamic environments. Instead, it would be preferable to have multiple inference models available to provide the flexibility to perform dynamic runtime adaptation for inference tasks. Designing a general framework for mobile deep inference faces the further obstacle of heterogeneous mobile hardware configuration. Therefore we would want access to a diverse set of models to adjust for the capabilities of each device.

Furthermore, selecting the best local model for individual applications is important given the increasing number of deep learning backed mobile applications and the constraints of on-device storage. Currently it is common for each application to package a single model since including more fine-tuned models would increase application size and worsen user experience. However, this deployment practice restricts runtime model flexibility. Instead it would be beneficial for an application to have access to a diverse set of models that expose tradeoffs between inference accuracy and speed.

Thus, delivering good mobile inference requires intelligent model selection algorithms that can dynamically trade-off among goals of accuracy and time within the varying constraints of mobile applications. In this paper, we propose a novel mobile deep inference platform, *MODI*, that centers on intelligent model management across mobile devices, edge servers and centralized servers. *MODI* follows a number of key design principles to increase the flexibility of runtime model selection. *MODI* solves the problems of what models to store on-device and determines what models to use for inference at runtime. We make the following contributions: (1) Design principles for improving inference time for heterogeneous mobile hardware and execution environments. (2) Preliminary results that demonstrate the feasibility

of proposed model management and dynamic adaptation solutions. (3) Open research questions that complement or further improve inference performance.

2 Background and Related Work

In recent years the use of deep learning models on mobile devices has exploded in popularity. A plethora of applications ranging from personal assistants [2] to applications such as Google Translate [9, 10] are leveraging *deep inference*, i.e., executing inference tasks via deep learning models,, to provide key application features. However, a major barrier for use of these models is their inherent complexity. Mobile devices have strong resource constraints such as processing power, battery life and network connectivity which are not present for traditional server-based inference engines. Below we discuss prior efforts to improve mobile deep inference performance and provide context.

Deep Inference Executions. There has been some development of moving existing deep learning frameworks to mobile devices [13, 20, 24]. However, these frameworks are generally too cumbersome to run efficiently on average devices [14] and thus lead to poor user experience. Instead, moving inference tasks off mobile devices to cloud-based inference engines [4, 5] means they are no longer limited by mobile hardware. This also makes the same inference feasible across all mobile devices regardless of hardware constraints [2]. However, this approach requires a fast and stable internet connection.

Model Optimizations. Another focus of previous work has been the redesign of models to use less powerful hardware. Compressing existing models through quantization [16] has decreased the memory footprints of models. Similarly, there have been moves towards new mobile focused model architectures [18, 19, 25]. These improvements generally come with some loss of accuracy [18]. *MODI* is designed to leverage these advancements in model optimizations by supporting dynamically chosen models at execution time. The details of our algorithm for dynamically choosing models is left to future work as the tradeoffs are complex.

Framework Redesign. Parallel to the redesign of models, frameworks are reworking their basic structure in order to improve mobile inference [7, 12] by moving away from GPU-only features and introducing mobile CPU optimizations. Our work builds on top of the existing mobile-specific frameworks by providing a thin layer of API that enables flexible inference executions.

Hardware Acceleration. Some modern mobile devices are equipped with specialized hardware [3, 17] that makes it possible to run powerful models on them. But they are the exception rather than the rule. Therefore, only a small subset of mobile users can benefit from the resulting accuracy and performance gains.

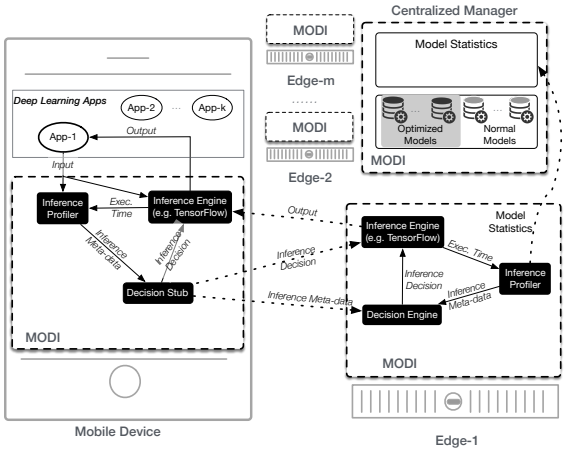


Figure 1: MODI system model. We depict the control and data flow between the mobile device and its edge server (Edge-1). Note that when the mobile user moves to a new location a different edge server Edge-i may be used.

On-device vs. edge server trade-offs. Previous studies [4, 6, 15] have proposed to offload computation intensive tasks to remote servers with the goals of improving response time and saving energy for resource-constrained devices. While utilizing edge servers can reduce the energy footprint and latency in many cases, specialized mobile hardware [14, 22, 23] has begun to close this gap. Such improvement in mobile hardwares makes it feasible to run on-device inference and necessitates the dynamic inference engine selection.

3 *MODI*: Mobile Deep Inference Platform

In this paper we propose a novel mobile deep inference platform, *MODI*, that is shown in Figure 1. *MODI* takes advantage of the diverse set of deep learning models across mobile devices and edge servers to dynamically deliver the best possible inferences. By ensuring that models are available locally, *MODI* can ensure critical functionality under network constraints while enabling fast, high quality inference on edge servers when needed.

To support this dynamic decision making, *MODI* follows three key design principles: (1) maximize local model storage to enable on-device inference for all possible cases, (2) monitor model metadata to enable intelligent model selection, and (3) dynamically select inference models based on environmental conditions. We discuss the implications of these principles in detail in Section 3.2 and present our preliminary results in Section 4.

System architecture: *MODI* is designed to be deployed across mobile devices, edge servers and a centralized

cloud manager. It consists of five major logical components. *Inference profilers* estimate the requirements and resources for each inference task, such as preferred accuracy, battery state and device connectivity. *Decision stubs* and *decision engines* work together to determine where to perform inference tasks based on the inference profile. Further, *decision engines* aggregate and analyze the decisions made by decision stubs and generate new model distribution plans that could further lead to changes in decision stubs. *Inference engines* use existing deep learning frameworks such as Caffe2 and TensorFlow [1, 20] and execute inference tasks using stored models. *Centralized Managers* are the master model repository of *MODI* and aggregate model usage and inference statistics across the system. Additionally, they push new models to mobile devices and edge servers.

These logical components are running on three physical entities. *Mobile devices* host inference profilers, decision stubs and inference engines, in addition to models as allowed by user-defined storage constraints. *MODI* tries to ensure mobile devices always have the most appropriate models, based on their specific resources and installed apps. *Edge servers* host inference engines, inference profilers and decision engines, as well as high-quality models. They use these models for inferences that cannot be serviced on mobile devices. Additionally edge servers collect statistics from devices and propagates this metadata to the centralized manager. *Central cloud servers* host a centralized manager to aggregate statistics about model usage. This allows the centralized manager to make global decisions regarding model quality and periodically update models stored elsewhere throughout *MODI*.

Assumptions: We assume that there are a diverse set of pre-trained models that are initially stored in a central cloud location, and that each of these models has a known storage requirement and provides a known inference accuracy [18]. Also, we assume that these models can be reused across applications or as smaller units in a large model. That is, a model m_A used in application A can be reused in other applications or can be combined with a second model m_B to form m_C. Previous work on transfer learning [18, 26, 29] supports this assumption.

3.1 A Motivating Example

As a motivating example consider a translation application, similar to Google Translate [9], that scans images and translates the text found in them. This application uses two models, the first which performs optical character recognition (OCR) to isolate the text in the image and the second to perform translation. These models each have different requirements and usage statistics. It is likely that the OCR model requires lower model accuracy as users can easily detect mistakes. Meanwhile, the translation model may be paired with other forms of input and thus be widely important.

The role of *MODI* in this app is twofold. First, *MODI* ensures that models for performing both OCR and translation tasks are available, enabling basic functionality in all cases and in case of network failure. Second, *MODI* can recognize that the translation task is more widely used and opt for higher accuracy. This application would run by requesting inference through *MODI* using a particular model. This would trigger an inference profile to be generated which is then passed along to the decision stub. If the local models are accurate enough and other resource constraints are met, the inference will be done locally. If either accuracy or other constraints cannot be satisfied locally, the inference will be routed to the edge server instead. The edge server then performs the inference and records the parameters of the request for analysis with its decision engine. The results of the decision engine are passed to the centralized manager who may decide to push a higher quality model to devices.

3.2 Design Overview

In this section, we explain the key design principles of *MODI* and discuss of a number of open questions.

3.2.1 Key principles

To accomplish its goals, *MODI* needs to be designed with a few key principles in mind. These are designed to ensure access to as wide a range of models as possible and to maximize scalability.

Maximize usage of on-device resource. Due to the relative unpredictability of network connections and the need for ensuring the availability of inference, we therefore aim to store as many models on device as possible. This manifests itself by exploring different model optimization and compression techniques.

Storage and analysis of metadata. To adapt to dynamic environments in mobile deep inferences, *MODI* needs to periodically re-evaluate both the model distribution plan as well as the runtime model selection algorithms. By analyzing metadata and usage statistics, *MODI* can more accurately determine which models are the most critical.

Dynamic runtime selection. To satisfy dynamic needs and constraints of mobile deep inference, *MODI* needs to decide on inference models at runtime based on current information and collected metadata.

3.2.2 Key Research Questions

We investigate a number of key questions that contribute to our vision of efficient mobile deep inference.

Which compression techniques are useful? Section 4.1 shows that there are variations in accuracy between compression methods. Therefore further study of compression and optimization techniques for reducing storage and their impact on performance is needed.

How to dynamically choose model for local inference? Different devices have different characteristics and predicting the most appropriate model is non-trivial. In Section 4.2 we examine inference differences between mobile devices of varying capacities.

When to leverage edge servers? *MODI* needs to ensure it is beneficial to offload inferences to edge servers when required. We analyze the feasibility of edge servers in Section 4.3

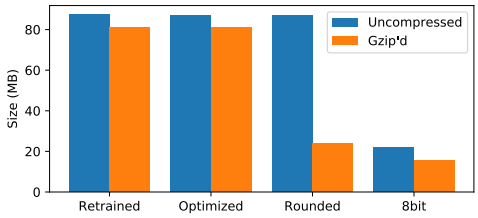
Which versions of models to store on a mobile device? Due to storage constraints and hardware heterogeneity, an exhaustive selection of models cannot be stored on device. We plan to develop an algorithm that accounts for accuracy, speed and energy requirements of models and determines which to store on-device.

4 Evaluation

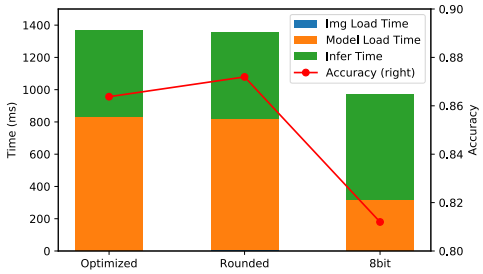
We present our preliminary results of *MODI* relating to on-device models and edge server hosted inference. In particular, we examine different model compression techniques and their impact on inference time and accuracy as well as the benefits of executing inference tasks on an edge server under varied network conditions.

Experiment Setup. Our inference experiments were conducted using the following Android mobile devices: a Google Pixel 2 (late 2017), a Moto G5 Plus (2016), a LG G3 (2014) and a Nexus 5 (2013), all running at least Android API 23 (Android Marshmallow). Our inference edge server was a `t2.medium` Amazon EC2 instance, located in Virginia data center [1]. Mobile devices are connected to this edge server through different networks including university WiFi, free public WiFi, residential WiFi and LTE. We developed an Android app for these devices to perform image classification tasks using a specified model and report the relevant metrics. For our model we used an InceptionV3 model [26] that had been retrained using a set of 3684 flower images [8]. We used a test dataset of 381 images for evaluating accuracy, defined as the number of correctly classified images. We then took the *retrained* model and optimized it for inference by removing training-only layers [8]. Finally, we applied quantization in the form of grouped weight-rounding and 8-bit quantization. These techniques focus on reducing model size, allowing the model to load faster. Note that efficient model loading is more useful for one-off mobile inference tasks than streaming inferences, such as video analysis, where longer model loading time can be amortized.

[1] When connecting through university WiFi, our Pixel 2 phone experiences an average network latency of 21ms to the server.



(a) Model storage size.



(b) Model inference speed and accuracy.

Figure 2: Comparisons of model compression techniques. Different models exhibit different requirements on mobile storage and experience various inference speed and accuracy. Overall, the 8-bit quantized model is superior in storage saving and inference speed with a slight 6% accuracy decrease. The retrained model is excluded from Figure 2(b) due to unsupported operations on mobile devices.

4.1 Impact of Model Compression

Compactly storing deep learning models is key to our vision of supporting a wide selection of on-device models. However, compression techniques generally trade-off inference accuracy for compression effectiveness. In this section, we first quantify the storage savings of four post-training compression techniques and then compare each technique's impact on inference performance.

Figure 2(a) compares the uncompressed and compressed model sizes. For each model, we plot the baseline uncompressed size (left bar) and the gzip version (right bar). As we can see, the 8-bit quantized model leads to the most storage saving of 75% regardless of gzip compression. In addition, the unquantized models (retrained and optimized) see only about 7% savings while the rounding quantized model sees a 72.6% storage reduction after being gzipped. Our observations suggest that both quantization and gzip compression can lead to significant storage savings, especially when combined.

Next, we compare the inference speed and accuracy of each model. Figure 2(b) shows the time taken by each model and its accuracy. It is important to note that ma-

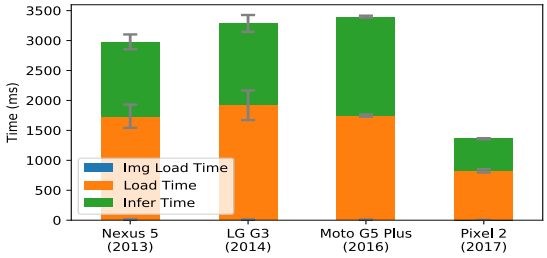


Figure 3: Comparisons of inference speed with the optimized model. Different mobile devices exhibit different time breakdown. The high-end Pixel 2 improves on-device inference speed by 2.3x compared to other devices.

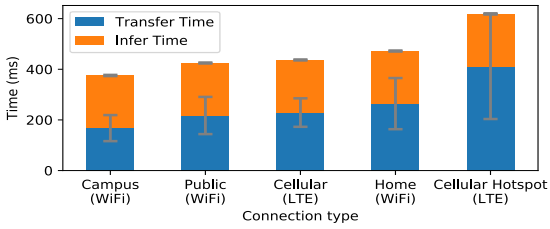


Figure 4: Comparisons of edge-server based inference under different mobile network conditions. Large variations are seen due to network conditions. Under poor network connectivity when using cellular hotspot, the transfer time almost doubled when compared to university WiFi.

jority of the inference time is in loading the model into memory and thus could be amortized over sequential mobile inferences. However, for one-off mobile inferences, the model loading time dominates the end-to-end inference time. The small 8-bit quantized model provides the fastest end-to-end response time with the lowest model loading time, but a slightly increased inference time. In our results, we do see a small accuracy increase for the rounding quantized model. But such observations are not common. This shows a case that would be very fortuitous in a mobile model and would be detected by the *MODI* through metadata tracking.

Summary: Model compression techniques have different impacts on model storage, inference speed and accuracy. MODI *could leverage these observations to carefully select techniques that provide different tradeoffs.*

4.2 Comparisons of Mobile Inferences

It is an important vision of *MODI* to support mobile devices with heterogeneous hardware capabilities by augmenting on-device models with edge-based hosting. To understand the capabilities of different devices in utilizing on-device models, in this section we compare the average end-to-end classification time that consists of image loading time, model loading time and inference time, as seen in Figure 3. As we can see, older or less capable devices (left three bars) take up to 3.5 seconds to present classification results to mobile users. However, the nearly 2 seconds of model load time could be amortized across multiple runs or reduced by using models with a smaller footprint.

Summary: Although newer mobile hardware is able to deliver acceptable user performance, other hardware would benefit from dynamically chosen inference models.

4.3 Benefits of Edge Model Hosting

As shown in Section 4.2, mobile-based inference does not always deliver good performance. Moreover, mobile-based inference is also constrained by the availability of on-device models and therefore is less flexible in making trade-offs between accuracy and speed. *MODI* supports an alternative inference option with edge-based model hosting. In this section, we evaluate the end-to-end classification time when using models hosted in edge servers with the optimized model pre-loaded. Figure 4 shows the average classification time using an edge server in a variety of network conditions. The majority of the classification time is network transfer time with up to 66.7% in the cellular hotspot case. Overall, the edge-server based classification ranges from 375ms to 600ms in a well-provisioned cloud server. When running the same classification task, Pixel 2 takes 536ms when the model is loaded which is on par with edge based inference.

Summary: Mobile-based inference only delivers acceptable performance for newer and high-end mobile devices while edge-based inference is a viable option even under poor network condition. MODI *can leverage this observation to dynamically select inference locations.*

5 Conclusion and Future Work

We introduced *MODI*, a platform for enabling efficient mobile deep inference through dynamic model selections. We demonstrated its feasibility through a number of experiments and explored open research questions.

Our future work will focus on designing model management and selection algorithms that are central to *MODI*. We plan to explore the impact of factors such as mobile device characteristics and network connectivity when making inference accuracy-speed tradeoffs.

6 Acknowledgements

We thank all the anonymous reviewers and our shepherd Marco Guazzone for their insightful comments, which improved the quality of this paper. This work is supported in part by NFS Grant CNS-1755659.

References

[1] ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I., HARP, A., IRVING, G., ISARD, M., JIA, Y., JOZEFOWICZ, R., KAISER, L., KUDLUR, M., LEVENBERG, J., MANÉ, D., MONGA, R., MOORE, S., MURRAY, D., OLAH, C., SCHUSTER, M., SHLENS, J., STEINER, B., SUTSKEVER, I., TALWAR, K., TUCKER, P., VANHOUCKE, V., VASUDEVAN, V., VIÉGAS, F., VINYALS, O., WARDEN, P., WATTENBERG, M., WICKE, M., YU, Y., AND ZHENG, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] APPLE INC. Deep Learning for Siris Voice: On-device Deep Mixture Density Networks for Hybrid Unit Selection Synthesis. `https://machinelearning.apple.com/2017/08/06/siri-voices.html`, 2017.

[3] APPLE INC. The future is here: iPhone X. `https://www.apple.com/newsroom/2017/09/the-future-is-here-iphone-x/`, 2017.

[4] CHUN, B.-G., IHM, S., MANIATIS, P., NAIK, M., AND PATTI, A. Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems* (New York, NY, USA, 2011), EuroSys '11, ACM, pp. 301–314.

[5] CUERVO, E., BALASUBRAMANIAN, A., CHO, D.-K., WOLMAN, A., SAROIU, S., CHANDRA, R., AND BAHL, P. Maui: Making smartphones last longer with code offload. In *ACM MobiSys 2010* (June 2010), Association for Computing Machinery, Inc.

[6] CUERVO, E., BALASUBRAMANIAN, A., CHO, D.-K., WOLMAN, A., SAROIU, S., CHANDRA, R., AND BAHL, P. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services* (2010), ACM, pp. 49–62.

[7] FACEBOOK. Delivering real-time AI in the palm of your hand. `https://code.facebook.com/posts/196146247499076/delivering-real-time-ai-in-the-palm-of-your-hand/`, 2016.

[8] GOOGLE. TensorFlow For Poets. `https://github.com/googlecodelabs/tensorflow-for-poets-2`.

[9] GOOGLE. How Google Translate squeezes deep learning onto a phone. `https://research.googleblog.com/2015/07/how-google-translate-squeezes-deep.html`, 2015.

[10] GOOGLE. Zero-Shot Translation with Googles Multilingual Neural Machine Translation System. `https://research.googleblog.com/2016/11/zero-shot-translation-with-googles.html`, 2016.

[11] GOOGLE. Pixel Visual Core: image processing and machine learning on Pixel 2. `https://www.blog.google/products/pixel/pixel-visual-core-image-processing-and-machine-learning-pixel-2/`, 2017.

[12] GOOGLE. Introduction to TensorFlow Lite. `https://www.tensorflow.org/mobile/tflite/`, 2018.

[13] GOOGLE. TensorFlow Mobile Overview. `https://www.tensorflow.org/mobile/`, 2018.

[14] GUO, T. Cloud-based or on-device: An empirical study of mobile deep inference. In *2018 IEEE International Conference on Cloud Engineering (IC2E'18)* (April 2018).

[15] GUO, T., SHENOY, P., RAMAKRISHNAN, K. K., AND GOPALAKRISHNAN, V. Latency-aware virtual desktops optimization in distributed clouds. *Multimedia Systems* (Mar 2017).

[16] HAN, S., MAO, H., AND DALLY, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR abs/1510.00149* (2015).

[17] HISILICON. Key Information About the Huawei Kirin 970. `http://www.hisilicon.com/en/Media-Center/News/Key-Information-About-the-Huawei-Kirin970`, 2017.

[18] HOWARD, A. G., ZHU, M., CHEN, B., KALENICHENKO, D., WANG, W., WEYAND, T., ANDREETTO, M., AND ADAM, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR abs/1704.04861* (2017).

[19] IANDOLA, F. N., MOSKEWICZ, M. W., ASHRAF, K., HAN, S., DALLY, W. J., AND KEUTZER, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR abs/1602.07360* (2016).

[20] JIA, Y., SHELHAMER, E., DONAHUE, J., KARAYEV, S., LONG, J., GIRSHICK, R., GUADARRAMA, S., AND DARRELL, T. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093* (2014).

[21] JOUPPI, N. P., YOUNG, C., PATIL, N., PATTERSON, D., AGRAWAL, G., BAJWA, R., BATES, S., BHATIA, S., BODEN, N., BORCHERS, A., BOYLE, R., CANTIN, P.-L., CHAO, C., CLARK, C., CORIELL, J., DALEY, M., DAU, M., DEAN, J., GELB, B., GHAEMMAGHAMI, T. V., GOTTIPATI, R., GULLAND, W., HAGMANN, R., HO, C. R., HOGBERG, D., HU, J., HUNDT, R., HURT, D., IBARZ, J., JAFFEY, A., JAWORSKI, A., KAPLAN, A., KHAITAN, H., KILLEBREW, D., KOCH, A., KUMAR, N., LACY, S., LAUDON, J., LAW, J., LE, D., LEARY, C., LIU, Z., LUCKE, K., LUNDIN, A., MACKEAN, G., MAGGIORE, A., MAHONY, M., MILLER, K., NAGARAJAN, R., NARAYANASWAMI, R., NI, R., NIX, K., NORRIE, T., OMERNICK, M., PENUKONDA, N., PHELPS, A., ROSS, J., ROSS, M., SALEK, A., SAMADIANI, E., SEVERN, C., SIZIKOV, G., SNELHAM, M., SOUTER, J., STEINBERG, D., SWING, A., TAN, M., THORSON, G., TIAN, B., TOMA, H., TUTTLE, E., VASUDEVAN, V., WALTER, R., WANG, W., WILCOX, E., AND YOON, D. H. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2017), ISCA '17, ACM, pp. 1–12.

[22] LANE, N. D., BHATTACHARYA, S., AND GEORGIEV, P. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)* (2016), pp. 1–12.

[23] NVIDIA. Gpus are driving energy efficiency across the computing industry, from phones to super computers. `http://www.nvidia.com/object/gcr-energy-efficiency.html`, Accessed on 2017.

[24] OSKOUEI, S. S. L., GOLESTANI, H., KACHUEE, M., HASHEMI, M., MOHAMMADZADE, H., AND GHIASI, S. Gpu-based acceleration of deep convolutional neural networks on mobile platforms. *CoRR abs/1511.07376* (2015).

[25] RASTEGARI, M., ORDONEZ, V., REDMON, J., AND FARHADI, A. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR abs/1603.05279* (2016).

[26] SZEGEDY, C., VANHOUCKE, V., IOFFE, S., SHLENS, J., AND WOJNA, Z. Rethinking the inception architecture for computer vision. *CoRR abs/1512.00567* (2015).

[27] VAN DEN OORD, A., DIELEMAN, S., AND SCHRAUWEN, B. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2643–2651.

[28] WU, Y., SCHUSTER, M., CHEN, Z., LE, Q. V., NOROUZI, M., MACHEREY, W., KRIKUN, M., CAO, Y., GAO, Q., MACHEREY, K., KLINGNER, J., SHAH, A., JOHNSON, M., LIU, X., KAISER, L., GOUWS, S., KATO, Y., KUDO, T., KAZAWA, H., STEVENS, K., KURIAN, G., PATIL, N., WANG, W., YOUNG, C., SMITH, J., RIESA, J., RUDNICK, A., VINYALS, O., CORRADO, G., HUGHES, M., AND DEAN, J. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR abs/1609.08144* (2016).

[29] YOSINSKI, J., CLUNE, J., BENGIO, Y., AND LIPSON, H. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2* (Cambridge, MA, USA, 2014), NIPS'14, MIT Press, pp. 3320–3328.

[30] ZOPH, B., VASUDEVAN, V., SHLENS, J., AND LE, Q. V. Learning transferable architectures for scalable image recognition. *CoRR abs/1707.07012* (2017).