Benchmarking Machine Learning Application on Edge Device

Connor Rawls
C00211180@louisiana.edu
University of Louisiana, Lafayette, Louisiana,

***Abstract*—Machine learning applications are becoming ever more prevalent in the world today. A trend in the design and functionality of these applications that can be observed is the move from all computations being done on a single monolithic server to a more decentralized architecture. With the emergence of cloud computing, it is easier than ever to provide machine learning applications to users who would otherwise be unable to utilize such powerful technology. Alongside these trends, mobile devices, such as cell-phones, are more commonplace than ever. These miniature computers can be found in the pocket of almost anyone, all over the world. Noting this fact, there is a large untapped potential in combining the compute logic of machine learning applications and the resource provisioning of abundant mobile devices. Utilizing transfer learning approaches to training, converting, and sending a machine learning model to a smaller device, the inference performance is compared between a representable edge device and a host machine. Unfortunately, the experiment was not successful in this regard. However, the model's performance was captured before and after quantizing the model, therefore, providing insight on the possible repercussions of transfer learning. Operating on the host machine, it was found that the original and transferred version of the model possessed an accuracy of 80.42% and 80.45%, respectively, as well as an average inference time of 4.52s and 5.23s, respectively.**

***Index Terms*—edge computing, quantization, transfer learning**

I. Introduction

In cloud computing architectures, small, outlying devices that send various data to the "central" compute server are considered edge devices. Central is considered loosely, as with distributed computing environments, the goal is to dissolve the traditional idea of a central computer. It is more common for these edge devices to relay data it has captured to the host (central) machine for it to be processed. The reason for this dynamic is that these edge devices are usually not resource abundant, providing poor performance for more intensive applications and programs. However, only using these devices as a means to collect data and disregarding any form of major computation potentially leaves the performance of this system dynamic sub-optimal. While the computational time on the edge device may prove greater than on the host machine, there is some data transferring latency that must be considered. In some cases, this latency is greater than the time it takes to make computations on the edge device, causing a performance decrease [1]. Keeping this factor in mind, it would be ideal to make strides in providing the edge devices with greater means to conduct application computations in-house, rather than outsourcing to some host machine. In consideration of the aforementioned realm of machine learning, these applications are inherently computationally expensive. Clearly, there would be a dichotomy between these types of applications and the hardware provided, leading to sub-par user experience or failure in mission critical environments. As the use for machine learning applications is becoming ever more popular, with such services as language translation, recommendation, and facial recognition [2], [3], [4], great consideration must be made on how to approach such a solution.

With this paper, the experiment of training an image classification machine learning model, quantizing the model, and comparing the inference performance between the two versions is examined. Additionally, the performance of the quantized version of the model is observed on a small-scale, mobile compute machine. This mobile device can be considered as the edge device in a distributed computing environment with the large-scale machine used to train the model as the host machine. The purpose of this experiment is to establish the performance comparison of a representable machine learning application on a host machine versus an edge device, further highlighting the possibility of a decreasing trend in computational time on smaller, mobile devices. The author of this paper theorizes that, given reasonable and practical approaches, machine learning models can perform inference on edge devices with a competitive accuracy and time as compared to traditionally performing inference on a host machine.

The contributions of this paper are the following:

- The profiling of training a model from scratch is studied and discussed.
- The accuracy of the model is examined once the model is quantized into a lighter version, representing any loss of performance due to transfer learning.
- The performance of a representable edge device conducting inference is theorized and compared to its host counterpart.

II. Related Work

There have been similar previous studies with the purpose of examining the intersection of machine learning and edge computing. One such work aims to provide knowledge on how machine learning could be adapted into the realm of edge computing and in which scenarios would benefit from such an environment [5]. Accordingly, this study sheds light

on some factors that should be considered in performing machine learning inference or training on edge devices, such as geographical location, and their effect on important metrics such as accuracy and latency.

Additionally, a study conducted in 2017 profiles the performance of machine learning applications with and without edge devices, capturing such metrics as accuracy and response time [6]. An interesting observation made in this study shows that user traffic and training data size play a significant role in the resulting performance of the application, with the edge device-included experiments being less impacted than the tests performing the computation without edge.

III. Background

A. Model Compression & Quantization

Machine learning models are often very large and require a complex system of operations to complete. With attempts to decrease the model's complexity and, in an adjacent manner, size, compression is an often used method after training has completed as a post-processing measure. Compression involves reducing not only the size of the model's parameters, but also the number of parameters in general. An important aspect that makes compression challenging is upholding the model's accuracy. There exists a tradeoff between compressing a model to achieve lower latency and require less resources while reducing its accuracy.

Quantization, as a form of compression, is an increasingly common approach in reducing the model's size. This method of compression focuses on reducing the size of the model's weights. The weights of a given fully-trained model will fall within a certain range of values. Quantization exploits this fact to uphold the weight distribution while decreasing their individual size. Typically, a model's weights are represented by floating point integers. With the TensorFlow library, two forms of quantization are offered: dynamic and full [7]. Dynamic quantization involves converting the weights from a floating point type to an 8-bit integer representation. In full quantization, all of the model's math is converted into 8-bit integers.

Methods of reducing the model's size and complexity are applicable to edge computing, as the smaller device would have less resources to compute with (such as disk space, memory, and compute power) and, in turn, would require all the help it could get in reducing the application's size and complexity. While a full-sized model might render an edge device worthless, a compressed model would allow the device to utilize applications that would otherwise only be available to larger, host machines.

B. Transfer Learning

Similar to the case that is studied in this paper, some environments operating machine learning inference applications have the compute resources to do so, however, do not have the resources to train the model within a reasonable amount of time. To combat this issue, transfer learning details how a model might be trained on one machine with more resources and then transferred to another, in our case, smaller machine

Hardware

	Raspberry Pi	Host Machine
CPU Model	Cortex-A72 (ARM v8) 64-bit	Intel i5-8600K 64-bit
CPU Speed	1.5 GHz	3.6 GHz
Memory	4 GB	4 GB
OS	Ubuntu 18.04	18.04

a)

Third-Party Python Libraries
tensorflow
tflite_runtime
numpy

b)

Fig. 1: System specifications showing a) the hardware details of both host and edge machines as well as b) the third-party Python libraries used to conduct the experiment.

to conduct a given task. With this experiment, this approach can be divided into two sections- the first being the training being done on the host machine and the second being the inference done on the edge device. Alongside using transfer learning as a solution for this challenge, it can also be utilized to partition the training process of a model. As an example, 75% of a model's training period can be conducted on one machine with the remaining 25% on another. This idea extends a machine learning application's use to a wider aggregation of machines.

IV. Methodology

During the training of the original model, the accuracy and loss was captured to profile its training performance. For each version of the model (original and compressed), the same experimental methodology and evaluation was used. The inference performance over 30 separate trials was captured with their values averaged. The metrics collected during testing to evaluate the models' performance were the time it took from starting the inference period until its completion and the model's accuracy. The hardware specifications of the machines as well as the third-party python libraries used in this experiment can be found in Fig. 1. As an additional note, the host machine lies within a virtual machine. While this might induce some latency during training, it is not believed that this causes any significant overhead during inference. To simulate the edge or mobile device, a Raspberry Pi 4B was used.

A. Model Architecture

The model used for this experiment was developed and trained using TensorFlow libraries [7]. The model was developed to be indicative of larger-scale, well-known image classification models [8]. The model possesses a total of 128,266 trainable parameters with its architecture displayed in Fig. 2. Originally, the model was trained without the dropout layer before layer 7. However, it was noticed that this resulted in a loss of accuracy and, therefore, was included into the model's final design. The model was trained for 100 epochs using the Fashion-MNIST dataset [9].

B. Dataset

The Fashion-MNIST dataset contains 60,000 training images and 10,000 testing images of various articles of clothing with such descriptors as t-shirt, ankle boot, and coat. Each image is of size 28 x 28 pixels. An example of these images

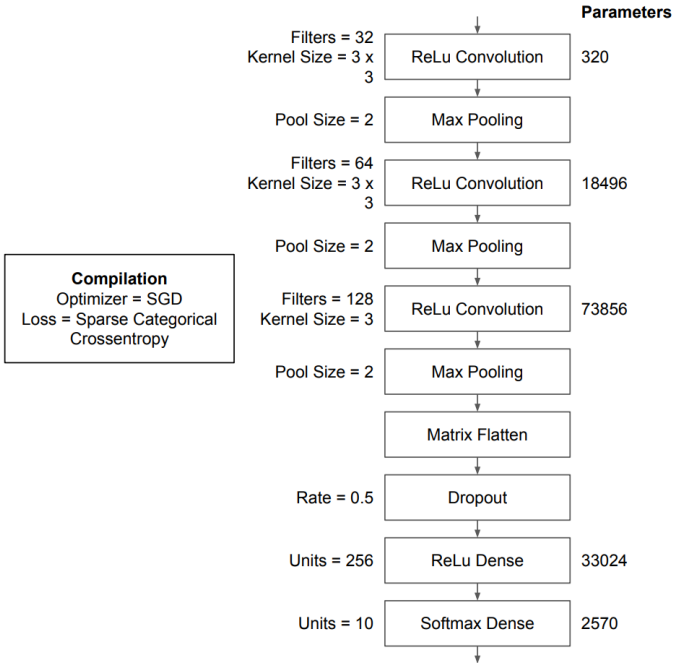

Fig. 2: Custom image classification model architecture.

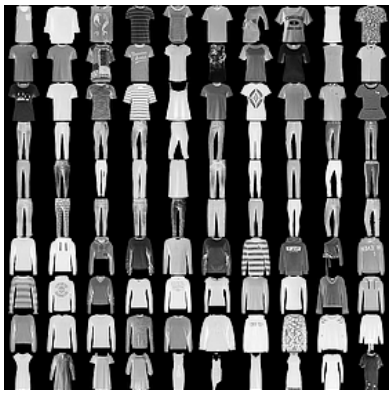

Fig. 3: Fashion-MNIST Dataset [9].

is shown in Fig. 3. The training set of images was partitioned into 50,000 images for the training subset and 10,000 images for the validation subset. It should be noted that for the experiment, the same set of training images was used between experimental runs. It would be ideal to utilize a different set of images between tests to capture the models' possible fluctuation in accuracy.

C. Model Conversion & Edge Inference

The original model was converted from a TensorFlow-based model to a TFLite-based one and compressed using dynamic quantization [10]. This lighter version of the model was tested first on the host machine before being transferred to the edge device. After the TFLite-based model was transferred from the host machine, inference on the edge device could then be observed using the same dataset as used for the original model's testing.

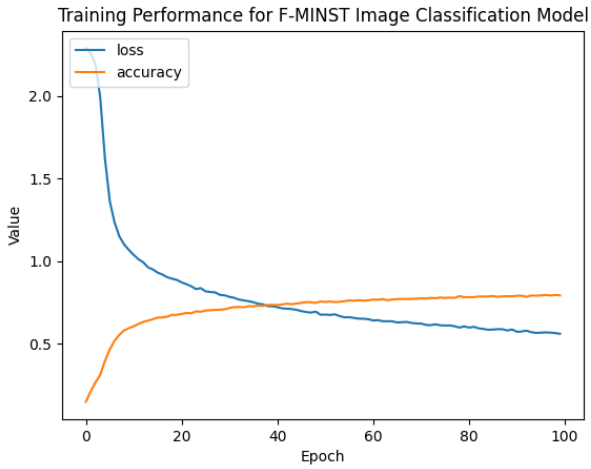

Fig. 4: Model training profile.

V. Experimental Results

While the initial part of the experiment involving training, testing, and converting the model was successful and provided insightful results, the latter part of the experiment with running the converted model on the edge device was unsuccessful. To compensate for the lack of data in this regard, thoughts and speculations will be given in its place. Consider these with a grain of salt, as they are unsophisticated in their substantiality.

A. Training Performance

The training profile can be observed in Fig. 4. One can view how the model's accuracy and loss begins to stagnate around the 100th epoch, indicating an optimal training period.

B. Performance of Models on Host Machine

The results from both the original model's performance and the converted, lighter model's performance on the host machine can be viewed in Fig. 5. It should be noted that the accuracy metric does not possess any deviation as the testing set remained the same through every run. The accuracy for the original and light models were found to be 80.42% and 80.45%, respectively. The marginal difference between these two values indicate that there is no significant loss in model accuracy when compressing the model to a quantized state. In fact, it can be observed that the compressed version of the model shows a 0.03% increase in accuracy.

The inference times for the original and light models were found to be 4.52s and 5.23s respectively. The 0.71 second delay induced by utilizing the quantized model for inference suggests that, given the edge device shows similar performance characteristics, there exists some threshold value that may determine if conducting inference on the edge device might prove more efficient than offloading the workload to a host machine. If offloading the workload induces more than a 0.71 second delay, then performing inference on the edge device should be strongly considered. If the offloading induced latency is less than this amount, then other considerations need

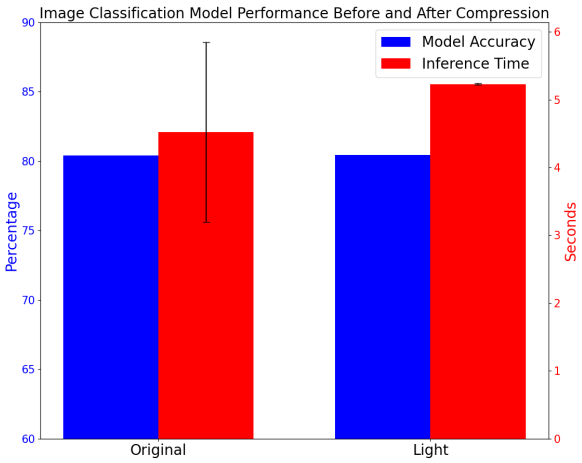

Fig. 5: Model performance.

to be made to further make computational decisions. Some of these factors might include limited mobile device battery life and geographical availability of the application service. The recognition and utilization of this slack may be decisive in choosing how to approach similar situations.

An interesting characteristic that was observed with the light model inference is the limited deviation in inference time (0.01s). However, the inference time deviation in the original model was found to be 1.32s, 29.20% of the average inference time. This deviation is significant and possibly suggests that the quantized version of the model is more robust than its uncompressed counterpart. In mission critical or user-facing environments, the application is expected to provide reliable and consistent behavior.

C. Theoretical Edge Performance

Due to the unsuccessful nature of this part of the experiment, this section is dedicated to speculating how inference should operate on the edge device as well as what the performance might look like, given the known hardware resources.

The light model was transferred from the host machine to the edge device with the intention of running inference tests. Picking up from this point, the model would be tested utilizing the same dataset as the one used with the tests on the host machine. However, there exists major conflicts between the architecture of the device used during the experimentation and the machine learning library used. Accordingly, operating machine learning applications on aarch64 architectures is in its experimental development cycle. As such, there exist many unforeseen bugs and glitches that make an already existing complex system even more difficult to troubleshoot. A possible avenue to get this environment operational may involve saving the input testing data as numpy arrays and reloading them on the edge system. This was attempted, however, due to time constraints, a working version could not be completed. Another possible solution to fix these errors may lie within making custom modifications to TensorFlow's Interpreter object. This module is required to be used as an alternative to the traditional interface used for various TF models.

Upon the consideration of the edge devices hardware in comparison to the host machine's, one can infer what the inference performance might look like. Being that machine learning applications are considered both I/O and computationally taxing, having greater memory and higher clock speed certainly help in performance. With these ideas in mind, it is shown that the edge device used in this experiment possesses a significantly weaker processor than the host machines. This would lead one to expect the inference time on the edge device to be greater than the inference time on the host machine.

VI. Future Works

A clear future for this work involves a functional version of performing inference on the edge device. Theorizing what the behavior might look like is not a strong basis to make conclusions for the possibility of adapting current machine learning technologies to edge devices.

In addition to a working version of the experiment, it would be ideal to implement a well known, large-scale model and dataset to perform tests with. While the work of this paper is representative, realistic environments may provide insight on unexpected behaviors and further validate the future of such studies. One such model that is highly recognized in the study of image classification today is ResNet [8]. Similarly, the work of this paper only considers a dataset of only 10 different labels. ImageNet is a popular large-scale dataset that contains many different image types that are complex, in comparison to the dataset used for this experimentation [11].

VII. Conclusion

This paper examined the performance of a trained-from-scratch image classification model before and after quantization, providing insight on the effects of transfer learning and the possibility of using such approaches in edge computing in distributed computing environments. While the goal of examining the behavior of the light version of the model on the edge device was not successful, some assumptions may be made on what the performance might look like. The potential for utilizing mobile devices in distributed systems is a largely untapped field that may prove very useful in the realm of machine learning applications and beyond.

References

[1] J. Zhang, X. Hu, Z. Ning, E. C.-H. Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633–2645, 2017.

[2] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[3] Z. Zhao, L. Hong, L. Wei, J. Chen, A. Nath, S. Andrews, A. Kumthekar, M. Sathiamoorthy, X. Yi, and E. Chi, "Recommending what video to watch next: a multitask ranking system," in *Proceedings of the 13th ACM Conference on Recommender Systems*, 2019, pp. 43–51.

[4] L. Stark, "Facial recognition, emotion and race in animated social media," *First Monday*, 2018.

[5] J. Chen and X. Ran, "Deep learning with edge computing: A review." *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[6] Y. Huang, X. Ma, X. Fan, J. Liu, and W. Gong, "When deep learning meets edge computing," in *2017 IEEE 25th international conference on network protocols (ICNP)*. IEEE, 2017, pp. 1–2.
[7] "Google tensorflow," https://github.com/tensorflow/tensorflow, accessed: 2012-12-02.
[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[9] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.
[10] "Google tensorflow lite," https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite, accessed: 2012-12-02.
[11] I. L. S. V. R. Challenge, "Olga russakovsky, jia deng, hao su, jonathan krause, sanjeev satheesh, sean ma, zhiheng huang, andrej karpathy, aditya khosla, michael bernstein, alexander c. berg, li fei-fei. 2014," *Computing Research Repository, Vol. abs/1409.0575*.