Optimizing Medical Image Classification Models for Edge Devices

Areeba Abid*, Priyanshu Sinha*, Aishwarya Harpale, Judy Gichoya, and Saptarshi Purkayastha

Abstract Machine learning algorithms for medical diagnostics often require resource-intensive environments to run, such as expensive cloud servers or high-end GPUs, making these models impractical for use in the field. We investigate the use of model quantization and GPU-acceleration for chest X-ray classification on edge devices. We employ 3 types of quantization (dynamic range, float-16, and full int8) which we tested on models trained on the Chest-XRay14 Dataset. We achieved a 2-4x reduction in model size, offset by small decreases in the mean AUC-ROC score of 0.0%-0.9%. On ARM architectures, integer quantization was shown to improve inference latency by up to 57%. However, we also observe significant increases in latency on x86 processors. GPU acceleration also improved inference latency, but this was outweighed by kernel launch overhead. We show that optimization of diagnostic models has the potential to expand their utility to day-to-day devices used by patients and healthcare workers; however, these improvements are context- and architecture- dependent and should be tested on the relevant devices before deployment in low-resource environments.

Areeba Abid*
Emory University School of Medicine, Atlanta, GA, USA, e-mail: areeba.abid@emory.address

Priyanshu Sinha*
Mentor Graphics India Pvt Ltd, Noida, India e-mail: priyanshu_sinha@outlook.com

Aishwarya Harpale
CakeSoft Technologies Pvt Ltd, Pune, Maharashtra, India

Judy Gichoya
Emory University School of Medicine, Atlanta, GA, USA

Saptarshi Purkayashta
Indiana University–Purdue University Indianapolis School of Informatics & Computing, Indianapolis, Indiana, USA e-mail: saptpurk@iupui.edu

*Authors contributed equally.

1 Background

Use the template *chapter.tex* together with the document class SVMono (monograph-type books) or SVMult (edited books) to style the various elements of your chapter content.

Instead of simply listing headings of different levels we recommend to let every heading be followed by at least a short passage of text. Further on please use the LaTeX automatism for all your cross-references and citations. And please note that the first line of text that follows a heading is not indented, whereas the first lines of all subsequent paragraphs are.

1.1 Motivation

Machine learning algorithms in healthcare show promise for alleviating disparities in access to healthcare, by providing automated diagnostic support in low-resource areas. However, these models are often developed (and limited to being run on) high-end hardware or cloud servers. To achieve equity in machine learning access and take advantage of widespread mobile access in limited resource settings, these models should be tested on edge devices, rather than being limited to high-powered servers. There are advantages and limitations of edge devices and high powered server architectures. Cloud servers have considerably more computational capacity and memory, but are expensive. Network bandwidth is finite, so downloading models or uploading data to servers is a resource-heavy task. On the other hand, edge devices are constrained in terms of memory and computing power, but are cheaper and not limited by network speed, making them more accessible in environments where financial and/or network resources are limited. In medical use cases, edge devices are additionally advantageous because they are not bound by HIPAA restrictions that limit the transfer of electronic protected health information (eHPI) to the cloud, as patient information does not need to leave the device in order to obtain a model inference [1]. Models can also be deployed to wearables in the context of chronic disease management, such as predicting blood glucose levels [2]. Due to the memory, inference latency, and privacy advantages offered by edge devices, deep learning models in healthcare gain much utility when optimized for deployment and execution in lower-resource environments.

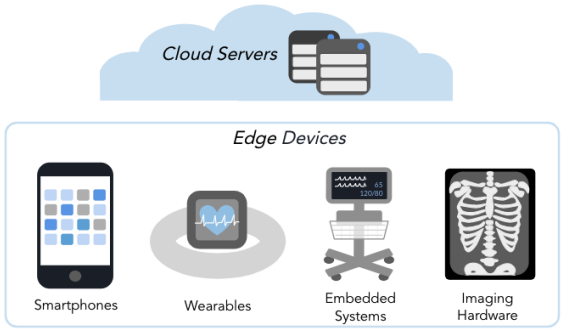
In this study, we evaluate the advantages and limitations of optimizing clinical machine learning models for edge devices. We study three metrics: model size, inference latency, and model accuracy as represented by area under the receiver operating characteristic (AUC-ROC) curve. We use radiology models trained on the NIH Chest-XRay14 Dataset and optimize these models by using 3 types of quantization: dynamic range, float-16, and full int8 quantization [3].

We hypothesize that if clinical models can be run faster and with reduced memory usage requirements on edge devices, models will be more suitable for deployment in limited-resource clinical settings for timely decision support.

1.2 Overview of Compression Techniques

Broadly, there are two common techniques for model compression for edge devices, *quantization* and *pruning* [6]. These methods reduce model complexity at a slight cost to accuracy, but offer improvements in computational speed and model

Fig. 1 Common machine learning contexts in healthcare.



size. This has the added benefit of reducing power consumption and bandwidth utilization.

Quantization is the reduction of model values to lower-bit representations. For example, a model trained with float32 precision can be compressed to use float16 or int8 precision for its weights, biases, and/or activations. This reduction in precision often has little impact on model accuracy, but reduces memory usage up to 4x (such as when reducing 32-bit floats to 8-bit integers). Computational speed also improves, as integer operations are generally faster than float operations on ARM CPUs [7]. Quantization can be implemented after model training (post-training quantization) or during training (quantization-aware training). Post-training quantization is faster to implement, but quantization-aware training typically offers better accuracy [8].

Connection Pruning is another common compression technique. In magnitude-based weight pruning, we drop low-weight connections between neural network layers. Models are represented by tensors; after connection pruning, the dropped connections are replaced by zeroes in the tensors. This results in sparse tensors for weight representations, which are easier to compress and reduce memory usage. As sparse representations contain zeroes, we can skip them while calculating an inference, reducing latency.

In this paper, we explore the advantages of quantization only, both post-training and quantization-aware training methods.

2 Method

2.1 Dataset

To demonstrate the efficacy of quantization for clinical use cases, we used the Chest-Xray14 Dataset, which consists of 112,120 X-ray images from 30,805 unique patients [3]. This dataset has been widely used to develop classification models for cardiopulmonary pathology. Each image in this dataset is annotated with labels from 14 pathology classes derived using text-mining from the associated radiology reports. The X-ray images can contain multiple pathologies, and each detected pathology is represented in a 1-by-14 vector as a positive class.

We randomly split the dataset into training (54,091 images), validation (23,183 images), and test (33,118 images) sets while ensuring that there was no patient overlap between each split. We performed pre-processing on each image by downscaling to 224×224 pixels.

2.2 Baseline FP32 Model

The floating point-32 model used as a baseline was Arevalo and Beltran's Chest X-Ray classification model ("Xrays-multi-dense121 0980aa"), developed using the DenseNet121 architecture [4]. This architecture consists of dense blocks followed by convolution and pooling layers. Each dense block receives feature maps from all the preceding layers and concatenates them to achieve a thinner and compact network. The model was initialized with weights pre-trained on the ImageNet dataset. An Adam optimizer is used to minimize the cost function starting with an initial learning rate of 0.001. Data generators were initialized with a batch size of 32.

Since each image can contain pathology in multiple classes, the output of the model is a 1x14 vector representing a probability score for each of the 14 pathology classes. The "No Finding" class is represented by a vector consisting of all zeroes.

We use this model as the baseline for size, inference latency, and accuracy comparison, and for generation of compressed models using post-training quantization.

2.3 Quantization of the Model

We implemented 3 types of quantization - Dynamic Range Post-Training Quantization, Float16 Post-Training Quantization, and Full Int8 Quantization-Aware Training. These compression methods were implemented using TensorFlow Model Optimization Toolkit [5].

Table 1 Summary of Model Quantization Techniques

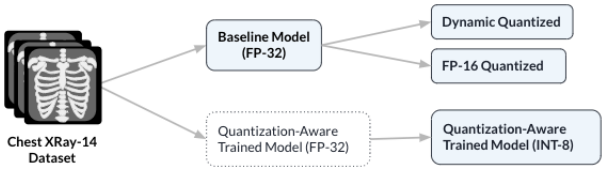
Compression Technique	Model Architecture
Dynamic Range Post-Training Quantization	Only weights are reduced to 8-bit integers. Activations are dynamically down-scaled to 8-bit at run-time for faster computation
FP16 Post-Training Quantization	Weights, biases, activations, and outputs are all reduced to 16-bit floats
Full Int8 Quantization-Aware Training	Weights, biases, and activations are all reduced to 8-bit ints. Input & output remain 32-bit floats.

Dynamic Range Post-Training Quantization: In dynamic range quantization, the model combines both floating point and 8-bit integer precision. The model weights and biases are scaled down from floating point to integer precision statistically by calculating the scale factor and zero point beforehand. Activations are dynamically quantized at run-time to 8-bit precision, so that computations between activations and weights can be performed in 8-bit precision. However, the outputs are then converted back and stored as floats. This reduces the latency of computation at inference close to that of fixed-point integer operations. This typically reduces the model size by about 75%.

Float16 Post-Training Quantization: We implemented Float-16 Quantization to reduce the full precision 32-bit floating point data to reduced precision. The weights, biases and activations are stored in 16-bits, and computations occur in the same format. Outputs are also stored in the 16-bit format. This typically reduces the model size by about 50%.

Full Int8 Quantization-Aware Training: During training of the initial FP32 model, quantization is emulated by down-scaling float precision so it matches 8-bit precision. The initial FP32 model is trained with floating point values, but during the forward pass of information, these values are converted to int8 and then back to FP32. This makes the float values less precise during training, which makes the model robust to quantization. After training, the model weights are quantized to int8, but since the weights were determined at lower precision, there is no additional loss of accuracy from the less-precise weights. Biases and activations are also reduced to int8, which does have some cost to accuracy, but allows for faster computation. This typically reduces the model size by about 75%.

Fig. 2 Development of quantized models.



2.4 Hardware Specifications and Costs

Edge devices come in a variety of hardware specifications, ranging from smartphones to an array of embedded devices and chips. To obtain representative results, we tested our compression techniques on a range of ARM and x86 architectures. The ARM devices used are the NVIDIA Jetson Nano, Raspberry Pi 3B+, Google Pixel, and Samsung Galaxy S10+. The x86 devices used are PC laptops using Intel x86 and AMD x86 processors. The costs of these devices range from less than 50 USD to over 1000 USD (Table 2). These costs can be compared to the costs of typical GPUs, both cloud and local (Table 3).

The choice of which devices to use was made based on CPU architecture, price range, and GPU availability. We include both ARM and x86 processors to investigate the effect of quantization on different architectures, which is critical to measure because these processors are optimized for computations on different data types: Arm processors have Integer computation accelerators, whereas x86 processors have floating point accelerators. We also examine the performance of the models over a spectrum of high-end and low-end devices, since a major application of edge computing is in resource-constrained environments. We include smartphone devices with optional GPU enabling, to allow for comparison of CPU- vs GPU- inference on a single device.

Table 2 Specifications of Test Devices: Architecture and Cost.

Device Name	ARM/x86	Specification	Price Range
NVIDIA Jetson Nano	ARM	ARMv8 Processor rev 1 (v8l), 4 GB RAM, 4 Core CPU	<100 USD
Raspberry Pi 3B+	ARM	4 Core CPU, 1 GB RAM, 1.4GHz processor, Broadcom Arm Cortex A53-architecture processor	<50 USD
Google Pixel (1st Gen)	ARM	Quad-core (2×2.15 GHz & 2×1.6 GHz) Kryo 64-bit ARMv8-A	650 USD
Samsung Galaxy S10+	ARM	CPU: Snapdragon 855; GPU: Adreno 640	999 USD
PC Laptop, Intel processor	x86	Intel Core i7-7820HQ CPU 2.90 GHz, 32 GB RAM, 8 Core	>1000 USD
PC Laptop, AMD processor	x86	Ryzen 7 3750H, 16 GB, 8 Core. Nvidia GeForce GTX 1660Ti with Max Q design GPU. VRAM: 6GB	>1000 USD

Table 3 Costs of Representative GPU Servers (Cloud and Local Examples)

Type of Server	Host / Service	Cost
Local GPU	ThinkStation Nvidia GeForce RTX2080 Super 8GB GDDR6 Graphics Card	1,100 USD [9]
Local GPU	Dell 16GB NVIDIA Tesla T4 GPU Graphic Card	3,904 USD [10]
Cloud GPU	Amazon Machine Learning	0.42 USD/hr + 0.10 USD per 1000 predictions [11]
Cloud GPU	Google Cloud, Basic-GPU	0.83 USD/hr [12]
Cloud GPU	Microsoft Azure, NC6 GPU	0.90 USD/hr [13]

2.5 Measuring Accuracy & Inference Latency

To measure the effect of model compression on *accuracy*, 33,118 test images from the ChestX-Ray 14 Dataset were used to evaluate the models and obtain AUC-ROC curves. For *inference latency*, each model was tested with 25 distinct pre-processed Chest X-Ray images. The model file was closed and reloaded between each image test. Two measurements were recorded; the total run time of the test for all 25 images, and the average inference latency (not including model and image file load time). The first measurement takes into account model load time and GPU kernel creation (if applicable), while the second measurement isolates inference latency only.

2.6 Code Repository

The code used to evaluate models is located at this repository:
https://github.com/areeba-a-abid/OptimizationEdgeDevices

3 Results and Discussion

The compression techniques used in this paper demonstrate varying degrees of improvement in model metrics for each device tested. The tables below describe

Table 4 Model Accuracy (AUROC) By Class. Note: Differences >0.05 between optimized models and baseline are bolded.

Class	Baseline FP32	Dynamic Quantization	Float16 Quantization	Quant-Aware Trained Int8
Atelectasis	0.78	0.78	0.78	0.75
Cardiomegaly	0.90	0.90	0.90	**0.71***
Consolidation	0.79	0.79	0.79	0.77
Edema	0.88	0.88	0.88	**0.79***
Effusion	0.87	0.87	0.87	0.84
Emphysema	0.88	0.88	0.88	**0.78***
Fibrosis	0.79	0.79	0.79	**0.73***
Hernia	0.83	0.83	0.83	**0.51***
Infiltration	0.71	0.70	0.71	**0.66***
Mass	0.82	0.82	0.82	**0.75***
Nodule	0.73	0.73	0.73	**0.65***
Pleural Thickening	0.77	0.77	0.77	**0.71***
Pneumonia	0.74	0.74	0.74	**0.66***
Pneumothorax	0.85	0.85	0.85	**0.77***
Mean AUC-ROC	0.81	0.81	0.81	**0.72***

Table 5 Size of Models

Architecture	Baseline Model	Dynamic Quant.	FP16 Quant.	QAT Int8
Model Size (MB)	27.9	7.3	14.1	7.4
Size Reduction	-	3.8x	2.0x	3.8x

the impact of model compression techniques on model size, model accuracy, and inference time.

3.1 Model Accuracy

Our results demonstrate that compression of deep radiology models is possible with very minor cost with respect to accuracy (Table 4). The Dynamically Quantized and FP16 Quantized models performed almost identically to the baseline model that they were derived from, with mean AUC-ROCs of 0.81 for both (the same as the mean AUC-ROC of the baseline FP32 model). The QAT Full Int8 model observed a decrease of 0.09 in mean AUC-ROC, with the 'Hernia' and 'Cardiomegaly' classes experiencing the biggest drop. In medical contexts, a class-by-class comparison is necessary to investigate where losses are most pronounced and for which types patients inferences should be used with extra caution. A reduction in sensitivity for high-mortality or costly conditions may lead to a larger impact on model utility as compared to the same percent reduction in the labeling-accuracy of benign conditions.

3.2 Model Size

The baseline FP32 model, which used 32-bit float representation for weights and activations, had a model size of 27.9 MB. FP16 Quantization reduced size by almost half, to 14.1 MB. By reducing representations to 8 bits, Dynamic and Int8 Quantization offered almost a 4x reduction, to 7.3 and 7.4 MB (Table 5).

3.3 Inference Latency

The degree of improvement in inference time was architecture-dependent. On ARM devices, integer computations are faster, so latency improved most with Dynamic Quantization and QAT Int8 Quantization. However, the more expensive smartphones (Google Pixel & Samsung Galaxy) did not show this improvement, because the Snapdragon CPUs do not support all quantized operations; investigating this is beyond the scope of this paper. The reduction in latency was most significant for the cheapest device (the Raspberry Pi), improving by 49-57% for these two integer quantization methods. The Jetson Nano and Samsung Galaxy demonstrated improvements between 25-49% (Table 6). FP16 quantization also offered a 13% reduction in latency on the Raspberry Pi, but increased latency on all other ARM devices. No significant improvement of FP16 models on ARM devices was expected, because the computations are still conducted using floats. The percent change for each model on ARM devices is shown in Figure 3).

For x86 devices, quantization methods that convert weights to integers actually increase latency by over 100x on the Intel processor and over 50x on the AMD processor. This is expected, as x86 devices are optimized for float computations. While integer quantization offers improvements for ARM devices, the dramatic effect on latency for x86 processors is a significant drawback to consider.

Investigating the effect of GPU on latency was done using the Samsung Galaxy S10+. When GPU is enabled, the time for inference per image is reduced for all models, but the overall run-time of the prediction increases. This is because setup time for the device's GPU kernel is expensive. Whether this trade-off is worthwhile is dependent on the number of images being passed into the model; beyond a certain number of inferences, the speedup of GPU surpasses the initial cost of setup.

The decision on if and how to optimize for edge devices is outlined in Figure 4.

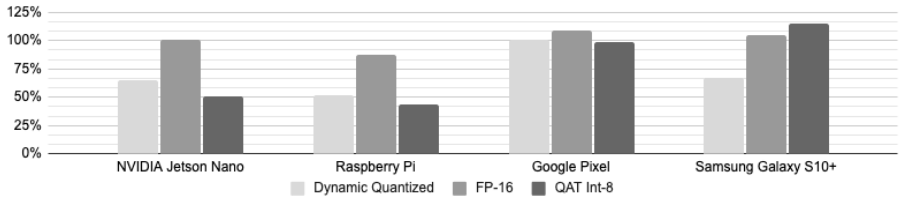


Fig. 3 Percent Change in Inference Latency for ARM Devices Compared to Baseline.

4 Conclusion

We find that model compression is an effective way to reduce model size by 2-4x with a minimal reduction in accuracy. This allows for a significant reduction in device cost and makes clinical models more accessible for a wider range of patients and healthcare providers, especially as machine learning models expand to a wide range of edge devices, such as smartphones, wearable technology, embedded devices, and imaging hardware. However, given the diversity of devices used in

Table 6 Inference Latency (ms) Per Image and Percent Change from Baseline Per Device

Architecture	Baseline Model	Dynamic Quant.	FP16 Quant.	QAT Int8
NVIDIA Jetson Nano	801	520 (↓35%)	806 (↑1%)	410 (↓49%)
Raspberry Pi 3B+	2340	1200 (↓49%)	2037 (↓13%)	1010 (↓57%)
Google Pixel	684	690 (↑1%)	741 (↑8%)	674 (↓1%)
Samsung Galaxy S10+	191	128 (↓33%)	201(↑5%)	220 (↑15%)
Intel x86	50	6070 (↑12040%)	50 (0%)	6937 (↑13774%)
AMD x86	100	5980 (↑5880%)	89 (↓11%)	6305 (↑6205%)

Table 7 Effect of GPU on Inference Latency and Total Run Time (ms) on Samsung Galaxy S10+

	Architecture	Baseline Model	Dynamic Quant.	FP16 Quant.	QAT Int8
Inference time only (per image)	CPU only: GPU-enabled:	191 124	128 125	201 120	220 124
Average run time (per image)	CPU only: GPU-enabled:	195 981	132 990	206 1002	223 1239

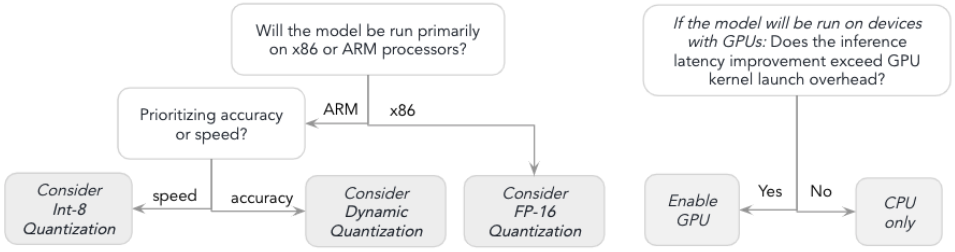


Fig. 4 To Optimize or Not to Optimize? Factoring in device types and priorities in optimization decisions.

medicine, it is important to note that the impact of model compression on inference latency varies depending on the architecture. Because x86 processors are optimized for float calculations, quantization to integers increases latency. Therefore, integer quantization methods are best suited for devices using ARM architectures.

Improvements in latency for x86 processors are demonstrated using FP16 models. Enabling GPU on higher end devices that have this option can also improve performance, but has the added cost of GPU kernel setup time. For example, in the context of radiology, a model that reads a single patient's X-ray images on-demand may be better off not utilizing GPU optimizations, but a use-case in which many patients' images are read at once may benefit from it.

As the availability of medical machine learning grows, we show that careful choices about model compression allow these advancements to be made more widely accessible, independent of access to high-cost devices or servers, but that the improvements offered by quantization are architecture- and context-dependent.

5 Future Work

In this paper, we used image classification as an example of the types of clinical models run on edge devices. Optimization for edge devices should also be explored in image segmentation problems, such as automated ultrasound segmentation; this is a rapidly growing area of model development and can be used at point-of-care on edge devices [14].

There are many more methods of model compression that should be further investigated. Quantization can be used to reduce model size further by reducing precision to 4-, 2-, or even 1-bit. Connection pruning, as discussed in section 1.2 can offer improvements independent of the type of processor used, and should be further explored.

Power usage is another important metric that was not explicitly investigated in this paper. Compression is expected to reduce the power usage of a model as a function of reduced run-time, but measuring and confirming this assumption was beyond the scope of this paper.

References

1. U.S. Department of Health and Human Services. "Guidance on HIPAA and Cloud Computing." (2020) https://bit.ly/3wyHFxD
2. Bhimireddy, Ananth, et al. "Blood Glucose Level Prediction as Time-Series Modeling using Sequence-to-Sequence Neural Networks." CEUR Workshop Proceedings. 2020.
3. X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers: ChestX-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases, in IEEE Conference on Computer Vision and Pattern Recognition (2017) https://arxiv.org/pdf/1705.02315v5.pdf
4. Arevalo, W., Beltran, J.: Xrays-multi-dense121 0980aa (2019). Retrieved from https://www.kaggle.com/willarevalo/xrays-multi-densenet121/data?select=weights.h5.
5. TensorFlow. Optimize machine learning models. Retrieved from https://www.tensorflow.org/model_optimization.
6. Massimo, M., et al. Edge Machine Learning for AI-Enabled IoT Devices: A Review. Sensors 2020, 20(9), 2533.
7. Ilin, Dmitry, et al. "Fast integer approximations in convolutional neural networks using layer-by-layer training." Ninth International Conference on Machine Vision (ICMV 2016). Vol. 10341. International Society for Optics and Photonics, 2017
8. Tensorflow. Quantization aware training. Retrieved from https://www.tensorflow.org/model_optimization/guide/quantization/training
9. ThinkStation Nvidia GeForce RTX2080 Super 8GB GDDR6 Graphics Card. Retrieved from: https://lnv.gy/3fJvQhj
10. Dell 16GB NVIDIA Tesla T4 GPU Graphic Card. https://dell.to/3fiw14m
11. AWS: Build a Machine Learning Model. Retrieved from: https://aws.amazon.com/getting-started/projects/build-machine-learning-model/services-costs/
12. Google Cloud AI Platform Pricing. Retrieved from: https://cloud.google.com/ai-platform/training/pricing
13. Azure Machine Learning pricing. Retrieved from: https://azure.microsoft.com/en-us/pricing/details/machine-learning/
14. AskariHemmat, M., et al: U-Net Fixed-Point Quantization for Medical Image Segmentation. MICCAI's Hardware Aware Learning for Medical Imaging and Computer Assisted Intervention (2019). https://arxiv.org/abs/1908.01073