Mobile-Optimized Real-Time Vessel Detection for Ultrasound-Guided Surgical Procedures

Mateusz Wolak, Fin Amin, Nancy DeLosa, Brian Telfer, Benjamin Roop, Lars Gjesteby
Human Health & Performance Systems, MIT Lincoln Laboratory, Lexington, MA, USA

***Abstract*—Non-compressible torso hemorrhage is the leading cause of potentially survivable fatalities in civilian and battlefield trauma. An insufficient number of trauma surgeons are expected to be available in future large-scale combat operations and natural disasters, creating a need for assistive technology to enable fast and accurate vascular access in pre-hospital environments. AI-GUIDE is a handheld surgical tool designed for emergency medical operations that combines an ultrasound probe with real-time image processing software, which controls robotic needle insertion components. Currently, AI-GUIDE relies on an external display/computer to perform the required computations. To reduce its size and weight, we investigate optimizations for mobile inference of the AI algorithms used in the AI-GUIDE prototype. Key optimizations include the use of mobile-optimized neural network models, quantization techniques, and leveraging a software development kit to harness portable hardware acceleration. We compare the trade-offs between speed and accuracy for different runtime configurations, quantization methods, and model sizes for two resource-conscious neural networks. We run our experiments on a smartphone as a reliable proxy for performance on the AI-GUIDE.**

***Index Terms*—real-time mobile object detection, portable ultrasound, vessel detection**

I. Introduction

To reduce pre-hospital deaths in civilian and battlefield trauma, life-saving interventions must be delivered by non-specialist medical providers. Specifically, vascular access to address internal hemorrhage requires significant training to find a deep blood vessel and insert a needle and catheter. The AI-Guided Ultrasound Intervention Device (AI-GUIDE) is a handheld prototype that combines portable ultrasound (US), AI algorithms, and handheld robotics to automate the most difficult steps in performing life-saving medical procedures [1]. The portable and robust nature of this device would help to prevent deaths under logistically-challenging casualty care conditions. As shown in Figure 1, the current AI-GUIDE paradigm is to transmit the US data to a standalone computer/display. However, due to the constraints of pre-hospital and austere environments, it is crucial to optimize the device for size, weight and power (SWaP) to make it easy to transport and operate. For this reason, we investigated various avenues for the next generation of handheld US. One way to achieve this is to move the AI processing of the ultrasound image feed from a standalone computer to a smartphone mounted to the handheld US probe. This requires the optimization of the AI algorithm to run efficiently in the compute and memory-constrained environment of a mobile system-on-chip (SoC). In our work, we refer to this new paradigm–with the AI processing on the smartphone–as the AI-GUIDE Mobile prototype.

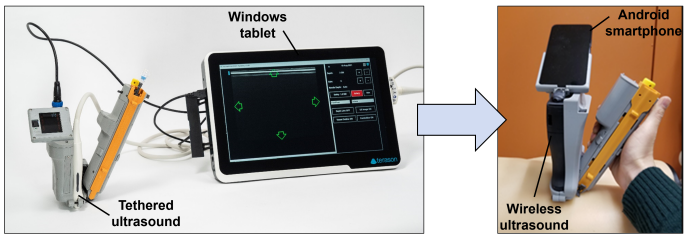


Fig. 1. The left image shows the original components and footprint of AI-GUIDE; note that the handheld device is connected to an external display/computer. The right image shows the AI-GUIDE Mobile prototype form factor with the AI computation moved to a smartphone integrated on the handheld device.

As a case study, this paper investigates the use of the Qualcomm Neural Processing SDK to optimize various object detection models for inference on a smartphone. Our motivation is to gain insight into the feasibility of an integrated AI-GUIDE Mobile platform for onboard AI computation. The goal is to present an empirical study across various accelerator/quantization combinations, which compares the tradeoffs between accuracy and inference speed.

II. Background and Related Work

In this section, we overview the existing research which guided our experiments and decisions. We predominantly focus on mobile platforms of biomedical AI in the context of *resource-conscious* deployment. Furthermore, we give an overview of the relevant AI-GUIDE inferencing pipeline. In our paper, we use the terms "patient" and "operator" to refer to the person receiving and giving medical care, respectively.

 Correspondence: Lars Gjesteby, MIT Lincoln Laboratory, Lars.Gjesteby@ll.mit.edu

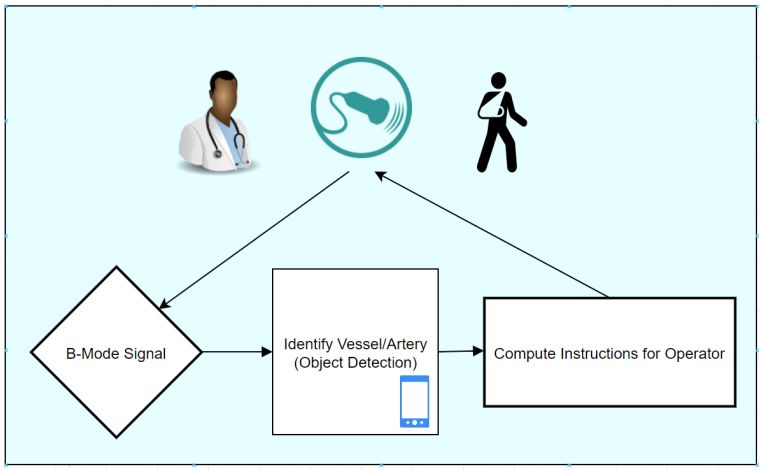


Fig. 2. The flow of information in between the ultrasound probe and the smartphone. The `Compute Instructions for Operator` component necessitates high object detection performance and low inference time from the `Object Detection` component. Otherwise, the operator would not receive timely/reliable guidance. Considering we target 30 FPS, we have a 33 ms budget for inference.

A. AI-GUIDE Inferencing Pipeline

Figure 2 describes the flow of information from the ultrasound data acquisition to the AI algorithm. Succinctly, the ultrasound device produces a B-Mode scan of the patient's anatomy. In order to guide the operator, the US images need to be segmented to detect the relevant landmarks, including blood vessels. As stated previously, the existing paradigm prescribes processing on a standalone device such as a tablet computer. Currently, the the standalone ultrasound device is a Terason uSmart 3200t (Burlington, MA), which has a volume of 146.25 in^3 and a weight of 4.85 lbs. Although this is portable, our aim is to reduce SWaP even further. A cellphone-sized device in lieu of the tablet computer could dramatically improve portability.

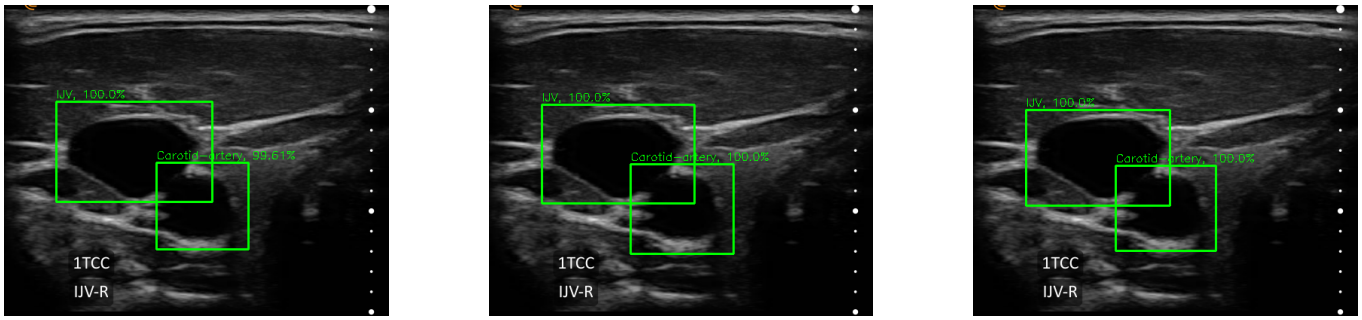
B. Handheld AI

Handheld AI refers to the implementation and optimization of artificial intelligence algorithms on portable devices, enabling real-time data processing and decision-making in resource-constrained environments. This technology is particularly valuable in scenarios where immediate analysis and response are critical, such as medical diagnostics, emergency response, and field operations. Although cloud computing can address some of these issues, the necessity of an internet connection introduces latency, reliability and security concerns. In the context of the AI-GUIDE Mobile prototype, handheld AI involves the integration of neural network models within a compact, mobile platform.

Work of this nature is not unprecedented [2]–[4]. There has been research on using smartphone apps to detect skin cancer with very high specificity [5], [6]. Other work has delved into handheld electrocardiograms [7]. Portable sonography has been an exciting frontier; for example, the *Butterfly* is a handheld ultrasound probe which connects to a phone [8]. Within the examples we surveyed, we noticed that mobile phones were the *de facto* device for performing at least some component of the processing/interfacing. For this reason, we perform our experiments on a smartphone; specifically, the Samsung Galaxy S22 is our representative device. We chose this Android device due to the onboard AI accelerators and its popularity among the prospective AI-GUIDE Mobile user group. We considered performing these experiments on the Nvidia Jetson, but even their smallest model consumes at least 5W, which exceeds our power limit [9].

C. Resource-Conscious Object Detection

For our experiments, we selected the MobileNetv2 SSDLite Model [10] and YOLOv8 [11]. MobileNetv2 builds upon the landmark MobileNetv1 [12], enhancing it with inverted residual connections and linear bottlenecks to improve the depth-wise separable convolutions introduced in the original work. MobileNetv1's depth-wise separable convolutions efficiently decompose standard convolutions into depth-wise and 1x1 point-wise convolutions, significantly reducing computational demands by 8-9x while maintaining comparable accuracy. MobileNetv2 refines this approach with inverted residual connections, which incorporate residual links between bottleneck layers within the convolutions, passing smaller tensors through these connections to reduce memory usage. This streamlined structure not only optimizes the computational graph layout in memory but also further diminishes memory requirements. Additionally, MobileNetv2 adapts the Single Shot Detector (SSD) [13], replacing standard convolutions with depth-wise separable convolutions, dubbed SSDLite, to further economize memory usage. Combined with linear activations in these bottleneck layers, MobileNetv2 achieves superior performance with substantially lower memory consumption than its predecessor, making it ideal for small SoC caches and optimal mobile performance. We opted for the MobileNetv2 SSDLite model due to its tailored optimization for mobile device performance and its established reputation as an industry-standard computer vision model for mobile applications. It



(a) Object Detection from MobileNetv2 SSD 320x320

(b) Object Detection from MobileNetv2 SSD 640x640

(c) Ground Truth

Fig. 3. Our experiments reveal promising object detection performance. The predictions in (a) and (b) were produced by the default quantization scheme using the Hexagon accelerator.

remains one of the fastest-to-inference models while maintaining competitive accuracy on various image classification tasks, as demonstrated in applications like systemic sclerosis skin identification [14]. Furthermore, this model is available with pre-trained weights on the ImageNet dataset, facilitating expedited training [15], [16].

YOLOv8 has been chosen for its high performance in real-time object detection tasks, owing to significant architecture enhancements that improve both detection speed and accuracy. The model builds on previous versions in the YOLO (You Only Look Once) model family, incorporating more efficient backbone networks and advanced feature pyramid networks to enhance multi-scale feature extraction. These improvements build on the signature capability of YOLO models to predict bounding boxes and class probabilities directly from full images in a single evaluation, which reduces computational cost and maintains high accuracy. Additionally, YOLOv8 training employs optimized loss functions and advanced optimization techniques, including stochastic gradient descent with momentum and weight decay, ensuring robust performance. Starting with pre-trained weights from the COCO 2017 dataset and fine-tuning on custom datasets further enhances its capability to adapt to specific detection tasks. In other works, it has achieved an mAP of 83.5, with 50 FPS throughput on object detection tasks in 1080p video [17]. YOLOv8's refined architecture makes it a strong candidate to compare against MobileNetv2.

III. Experimental Setup

In this section we describe the process of fine-tuning the pre-trained MobileNetv2 SSD and YOLOv8 models using our ultrasound image dataset, converting it to the Qualcomm `.dlc` (deep learning container) format, quantizing the trained model, and deploying it to the Samsung S22 smartphone using an Android app with Qualcomm's software development kit (SDK).

A. Ultrasound Dataset

The dataset we used to fine-tune the model contained scans of ultrasound data captured from the necks of 16 normotensive human subjects using a Clarius wireless L7A ultrasound device (Vancouver, BC, Canada). All data were collected by Massachusetts General Hospital and MIT Lincoln Laboratory with IRB approval. These scans are represented as individual .png images of sequential frames captured at 30 FPS. The frames were of varying sizes but were all resized to 640x640 pixels and normalized to the range $[-1.0, 1.0]$. The labels were provided by trained human annotators on each frame with bounding boxes for landmarks of interest, including the carotid artery, internal jugular vein (IJV), and branching vessels. The carotid artery is a major blood vessel in the neck that supplies blood to the brain, neck, and face. It is essential for delivering oxygenated blood to the cerebral regions. The internal jugular vein is a large vein that carries deoxygenated blood from the brain, face, and neck, back to the heart. It runs alongside the carotid artery. The compressed IJV was also labeled as a separate class to serve as a surrogate for hypotensive conditions [18], [19]. In total, the dataset contains 20,900 frames, with 13,809 used for training, 3,452 used for validation, and 3,639 used for testing, where each subject's data is kept together in the same dataset, meaning we test on completely unseen subjects.

B. Model Training

In order to achieve better results with shorter training times and make full use of our relatively limited dataset without overfitting, we initialized our MobileNetv2 SSDLite model with pre-trained weights provided by the TensorFlow Model Garden [16]. With these pre-trained weights, the MobileNetv2 image classifier backbone was trained on the ImageNet dataset, while the SSDLite object detection layers were trained on the COCO 2017 dataset [20]. We used this pretrained checkpoint to initialize our model's learnable parameters, then trained it on our ultrasound dataset. This was done in Tensorflow 2 for 217 epochs with a batch size of 80, on two NVIDIA V100 GPUs. We trained two input sizes of MobileNetv2 using this method: 640x640 and 320x320 pixels. For each resolution there is a separate model and training run. For YOLOv8, we used the model pretrained on the same COCO dataset. Note that unlike MobileNetv2, we used a PyTorch backend to train

YOLO for 100 epochs with a batch size of 16. Note that the loss functions for both YOLOv8 and MobileNetv2 are quite sophisticated; we direct readers to their respective papers for more details [13], [17].

C. Model Evaluation

After the model was trained, its performance was evaluated on the test set using one NVIDIA V100 GPU. The metric used was area under the precision-recall curve (AuC) derived from intersection over union (IoU) scores for the bounding boxes. IoU is computed by dividing the area where the predicted and ground truth bounding box overlap by the total area of both boxes. A true positive (TP) is recorded when the IoU between a predicted and ground truth bounding box is over 0.5, and a false negative (FN) is recorded when no predicted bounding box has an IoU over 0.5 for a given ground truth box. A false positive (FP) is recorded when no ground truth label exists for a given predicted bounding box with an IoU over 0.5, where one label cannot be re-used for multiple predictions. True negatives (TN) are not possible due to the nature of object detection. The AuC is computed by calculating the precision (TP/(TP+FP)) and recall (TP/(TP+FN)) at 11 decision thresholds, ranging from 0.0 to 1.0 in steps of 0.1. The decision threshold determines above what confidence value bounding boxes are considered for the calculation. We then plot precision against recall, generating a curve between (0.0, 1.0) and (1.0, 0.0), and calculate the AuC to compute the final metric.

D. Model Conversion

With our particular implementation of MobileNetSSDv2, designed to be re-used for many target environments, the model must first be converted from a TF SavedModel to a TFLite file. This is done through a script provided with the pre-trained weights on the TF Model Garden, and includes only the appropriate model endpoints for mobile inference. Next, the TFLite model must be converted to Qualcomm's proprietary `.dlc` format for use with their Neural Processing SDK on the S22's SoC. The compiler portion of the SDK, which handles the conversion from various saved model formats to `.dlc`, must be run on Linux for the full set of features. In our experiment, we ran the toolchain on an Ubuntu 20.04 LTS instance within WSL 2 on a Windows 10 machine. This conversion is handled by the `snpe-tflite-to-dlc` tool. Next, Qualcomm provides a number of quantization options, described in the next subsection. Quantization is applied to the DLC file using the `snpe-dlc-quant` tool. Finally, the computational graph is prepared for use with the specific Hexagon architecture on the Snapdragon 8 Gen 1 SoC, using the `snpe-dlc-graph-prepare` tool.

E. Quantization

Ordinarily, the numbers that make up the weights, biases, and data within a neural network are 32-bit floats, the standard representation for a floating-point number. Quantization seeks to reduce the size of these values, "squeezing" the 32-bit float into smaller, usually integer representations, like 8-bit integers. Intuitively, the learned parameters should stay approximately the same, and the model's outputs, as long as the quantization is "undone", should not change much, leading to a more favorable tradeoff between accuracy and speed. In practice, this has been shown to be a very effective method for increasing the speed of a network and has become standard practice for deploying models to mobile SoCs.

Qualcomm provides a few different quantization algorithms in the Neural Processing SDK, which are aware of the hardware runtime targets on Qualcomm's SoCs through the `snpe-dlc-quant` tool. These quantization options are: default, enhanced, adjusted weights, cross-layer equalization (CLE) [21], and symmetric. The default option uses the min/max of the data sample to map floats into 8-bit ints and performs an adjustment to ensure a minimum range and that 0.0 is exactly quantizable. The enhanced quantizer, per Qualcomm's documentation, uses a proprietary algorithm to determine a set of parameters that produces better performance. Adjusted mode excludes long tails of the data, which may help in some cases. CLE is a technique developed by Qualcomm for the convolutional layers used in vision models [21], using a scale-equivariance property of the activation functions. Qualcomm claims this can mitigate large drops in performance with quantization. Finally, symmetric quantization is used to ensure the negative and positive ranges of the quantization mapping are the same. These options can also be combined with one another using the tool.

F. Android Runtimes

Once the `.dlc` file has been generated, it must be prepared for a particular set of Qualcomm SoCs, which is done using the `snpe-dlc-graph-prepare` tool and a list of target SoCs for the model. This is mainly for use with the Hexagon DSP accelerator, which has a different architecture in different SoC generations. Finally, the prepared `.dlc` file can be used with the Neural Processing SDK's Android library to run model inference on a mobile device with a Qualcomm SoC. The SDK provides a number of runtimes, which execute the model on different hardware elements of the SoC. These are: CPU (central processing unit), CPU Quantized, GPU (graphics processing unit), GPU FP16, DSP (digital signal processor), and AIP (AI processor). The CPU mode runs only non-quantized models in 32-bit floating point on the CPU. CPU Quantized mode allows quantized models with 8-bit integer weights to run on the CPU, though it is generally not optimized for this. The GPU also runs only non-quantized models, and uses 32-bit floating point representations. The special FP16 mode automatically converts the unquantized model to use 16-bit floating point weights, and can leverage hardware optimizations for this representation. In general, the GPU can have more optimized hardware for matrix multiplication. The DSP and AIP targets are for different generations of the Hexagon accelerator built into Snapdragon SoCs. For our SoC (Snapdragon 8 Gen 1, or SM8450), this is called the DSP, and contains hardware optimizations for 8-bit integer

TABLE I

PERFORMANCE TRADEOFF FOR 320X320 RESOLUTION MODELS. THE BEST-PERFORMING VALUE IS BOLDED IN EACH METRIC COLUMN. INFERENCE SPEEDS WHICH ARE TOO SLOW FOR 30 HZ ARE WRITTEN IN RED. N/A VALUES INDICATE UNREPORTED RESULTS.

Model	Quantization	Runtime	Speed (ms)	AuC	Model	Quantization	Runtime	Speed (ms)	AuC
MobileNet v2 SSD-320x320	None	CPU	62.6	0.7980	YOLO v8-320x320	None	CPU	61.9	0.8902
	None	GPU	23.0	0.7960		None	GPU	18.8	**0.9237**
	None	GPU_FP16	20.7	0.7980		None	GPU_FP16	17.3	0.9117
	Default	CPU_QUANT	17.7	0.7990		Default	CPU_QUANT	16.2	N/A
	Default	DSP	3.2	0.7950		Default	DSP	**3.7**	N/A
	Enhanced	CPU_QUANT	18.2	0.7980		Enhanced	CPU_QUANT	16.2	N/A
	Enhanced	DSP	3.2	0.7960		Enhanced	DSP	3.8	N/A
	Adjusted	CPU_QUANT	18.2	0.8000		Adjusted	CPU_QUANT	16.2	N/A
	Adjusted	DSP	3.1	0.7990		Adjusted	DSP	3.9	N/A
	Override	CPU_QUANT	18.1	0.7990		Override	CPU_QUANT	16.2	N/A
	Override	DSP	3.2	0.7960		Override	DSP	**3.7**	N/A
	Symmetric	CPU_QUANT	29.1	**0.8220**		Symmetric	CPU_QUANT	15.8	N/A
	Symmetric	DSP	**3.0**	0.8160		Symmetric	DSP	3.9	N/A

TABLE II

PERFORMANCE TRADEOFF FOR 640X640 RESOLUTION MODELS. THE BEST-PERFORMING VALUE IS BOLDED IN EACH METRIC COLUMN. INFERENCE SPEEDS WHICH ARE TOO SLOW FOR 30 HZ ARE WRITTEN IN RED. N/A VALUES INDICATE UNREPORTED RESULTS.

Model	Quantization	Runtime	Speed (ms)	AuC	Model	Quantization	Runtime	Speed (ms)	AuC
MobileNet v2 SSD-640x640	None	CPU	223.7	0.8245	YOLO v8-640x640	None	CPU	209.4	**0.9159**
	None	GPU	57.6	0.8335		None	GPU	48.4	0.9078
	None	GPU_FP16	55.2	**0.8337**		None	GPU_FP16	41.2	0.9058
	Default	CPU_QUANT	53.6	0.8110		Default	CPU_QUANT	54.9	N/A
	Default	DSP	7.3	0.8105		Default	DSP	11.0	N/A
	Enhanced	CPU_QUANT	54.1	0.7799		Enhanced	CPU_QUANT	55.0	N/A
	Enhanced	DSP	**7.2**	0.7847		Enhanced	DSP	11.1	N/A
	Adjusted	CPU_QUANT	53.6	0.7797		Adjusted	CPU_QUANT	55.2	N/A
	Adjusted	DSP	7.4	0.7878		Adjusted	DSP	10.9	N/A
	Override	CPU_QUANT	53.5	0.8101		Override	CPU_QUANT	55.6	N/A
	Override	DSP	7.4	0.8108		Override	DSP	**10.8**	N/A
	Symmetric	CPU_QUANT	111.2	0.8104		Symmetric	CPU_QUANT	55.1	N/A
	Symmetric	DSP	7.4	0.8106		Symmetric	DSP	11.1	N/A

matrix multiplication, and is used with the quantized model (DSP Arch v69). Our SoC does not have an AIP, so this option was not used; however, in different processors this refers to a coprocessor further optimized for AI models. The models are run on the Android device through Qualcomm's Android library. The Android app must handle populating the input tensors and interpreting the output tensors. This can involve de-quantizing the results, and correctly mapping them to bounding boxes and classes.

G. Benchmarking

For our experiments, we created an Android app, which used Qualcomm's Android runtime library to run experiments using our converted and quantized `.dlc` model files. This was a simple app, which allowed us to select a model file, dataset folder, and runtime target. It then performed inference, collected timing data, and saved the model's predictions to local storage, so we could evaluate the model's accuracy. For each experiment, we tested a different combination of runtime and quantization options on the test set. We tested 5 runtime options: CPU, GPU, GPU_FP16, CPU_QUANT, and DSP, along with 6 quantization options: none, default, enhanced, adjusted, override, and symmetric. The override option refers to an optional override flag to ignore and overwrite any quantization or optimizations already present on the model before conversion. We were unable to obtain results for Qualcomm's CLE quantization, as it would cause the model to output very large numbers in the output tensors, which lead to a final accuracy of almost zero. We believe this is because the CLE optimizer is only designed to work with certain layer types and activation functions, and our models were not fully compatible.

Additionally, the CPU, GPU, and GPU_FP16 runtimes do not support quantization, while CPU_QUANT and DSP only support quantized models, so not all combinations were tested. For each experiment, we ran inference on the test set 3 times and took the median result to ensure random variations due to background processes, thermal throttling, or other factors did not affect our results. Timing results were obtained through the Android app, using the `System.currentTimeMillis()` function to measure how much time the model inference function call took for each image, and then averaging the result. When running an experiment, the app also output the predicted bounding boxes as `.txt` files. Detection performance was by exporting the predicted bounding boxes to our compute cluster, and comparing them to our ground truth results using our AuC metric.

IV. Results and Discussion

Our experiments have revealed insightful performance characteristics across various models and quantization schemes. For the 320x320 resolution models, as shown in Table I, the best performance in terms of speed was achieved with the DSP runtime. Specifically, the symmetric quantization on DSP yielded the fastest inference time of 3.0 ms for MobileNet and 3.7 ms for YOLO. The highest object detection performance for the MobileNet model was achieved with symmetric quantization on the CPU, with an AuC of 0.8220, whereas YOLO attained its highest object detection performance using the GPU runtime without quantization, with an AuC of 0.9237. Interestingly, while the enhanced quantization did not significantly improve performance, symmetric quantization provided a benefit in object detection performance for MobileNet models. This finding is counter intuitive as typically, quantized models are outperformed by their unquantized counterparts. We surmise that this discrepancy was caused by label noise on the test set, and that the use of "improved" quantization algorithms does not have a major impact on accuracy.

For the 640x640 resolution models, as shown in Table II, the symmetric quantization combined with the DSP runtime offered a balanced tradeoff between speed and object detection performance for MobileNet, with an inference time of 7.4 ms and an AuC of 0.8106. YOLO models performed best in terms of accuracy using the GPU runtime without quantization, with an AuC of 0.9159. However, the DSP runtime with default quantization achieved an inference time of 11.0 ms, which is within acceptable limits for many real-time applications.

Although our MobileNet retained comparable detection performance when using quantization, our experiments found the object detection performance of quantized YOLO models to be lacking. After analyzing the outputs from quantized YOLO models, we noticed that the class probabilities were the same low (< 0.01) float value for all classes and predictions within each image. We surmise this is due to the logits in the final layer having small magnitude. This in turn could cause the quantization process to map these logits to the same value. As such, we did not include these results, as we do not believe they are representative of the potential quantized performance of YOLOv8. Additionally, the quantization tools and runtime from Qualcomm and scripts from Ultralytics were configured and used exactly according to standard guidelines, but subtle changes to options in these tools could cause sub-optimal performance. Given the intricate nature of deep learning model optimization and deployment, even minor deviations from the optimal configuration can lead to significant performance variations. We sought to follow the standard guidelines and best practices, but there is a possibility of errors in our implementation or experimental setup for YOLO. Future work should delve deeper into available optimizations and export options within the Qualcomm and Ultralytics tools, to preserve YOLOv8 performance during quantization. An interesting connection can be drawn between our work and [22]. Similar to ours, their research delves into inferencing under extreme mass and energy constraints–however the context of their experiments is for space applications.

The DSP runtime demonstrated significant advantages in speed, making it suitable for real-time applications where inference time is critical. Our study underscores the efficacy of quantization and the DSP runtime for mobile deep learning inference, greatly increasing throughput with only a minor penalty to accuracy. For the AI-GUIDE Mobile prototype, the optimal configuration would bethe 640x640 symmetric quantized MobileNet model on the DSP, which outperformed other configurations in both metrics. Additionally, YOLOv8 showed promising quantized inference speed and unquantized accuracy, making optimization of quantized YOLOv8 models an important future step.

V. Summary and Future Work

In summary, this work analyzed multiple accelerators and quantization methods for vessel detection algorithms on ultrasound images, with the goal of achieving 30 FPS speed at high object detection performance. The default quantization on the DSP provided excellent speedup for both the MobileNetv2 SSDLite and YOLOv8 models, while maintaining high precision-recall AuC for MobileNetv2, particularly at 640x640 image resolution.

In the future, there are a number of investigations to be carried out. First, a larger set of computer vision models could be analyzed, including the latest iteration of the YOLO family, mobile-optimized vision transformer models, and video-based object detection models, such as Robust and Efficient Post-Processing [23] and YOLOV [24]. Given the strong speedup of the DSP with MobileNetSSDv2 and YOLOv8, these slightly more complex models could yield better accuracy, while still achieving adequate inference speed. Next, a wider range of mobile and low-power SoCs and accelerators could be evaluated, including the latest flagship Snapdragon SoC, stand-alone development boards, or the NPU built into the new Snapdragon X Elite processor. Furthermore, Qualcomm's Userbuffer memory interface could be added to the benchmark app. Qualcomm claims using this special memory interface in the Android app saves a memory copy operation when loading inputs and outputs to and from the model, and this could improve overall performance. Finally a more optimal configuration of the Qualcomm and Ultralytics tools, along with other optimizations, such as quantization-aware training [25] should be investigated to make YOLOv8 suitable for the AI-GUIDE Mobile prototype.

Acknowledgment

The authors would like to acknowledge the MIT Lincoln Laboratory Supercomputing Center (LLSC) for their support of high performance computing tasks.

References

[1] L. J. Brattain, T. T. Pierce, L. A. Gjesteby, M. R. Johnson, N. D. DeLosa, J. S. Werblin, J. F. Gupta, A. Ozturk, X. Wang, Q. Li *et al.*, "AI-enabled, ultrasound-guided handheld robotic device for femoral vascular access," *Biosensors*, vol. 11, no. 12, p. 522, 2021.

[2] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Ai and ml accelerator survey and trends," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2022, pp. 1–10.

[3] N. A. Switz, M. V. D'Ambrosio, and D. A. Fletcher, "Low-cost mobile phone microscopy with a reversed mobile phone camera lens," *PloS one*, vol. 9, no. 5, p. e95330, 2014.

[4] J. T. Coulibaly, M. Ouattara, M. V. D'Ambrosio, D. A. Fletcher, J. Keiser, J. Utzinger, E. K. N'Goran, J. R. Andrews, and I. I. Bogoch, "Accuracy of mobile phone and handheld light microscopy for the diagnosis of schistosomiasis and intestinal protozoa infections in côte d'ivoire," *PLoS neglected tropical diseases*, vol. 10, no. 6, p. e0004768, 2016.

[5] T. M. de Carvalho, E. Noels, M. Wakkee, A. Udrea, and T. Nijsten, "Development of smartphone apps for skin cancer risk assessment: progress and promise," *JMIR Dermatology*, vol. 2, no. 1, p. e13376, 2019.

[6] A. Udrea, G. Mitra, D. Costea, E. Noels, M. Wakkee, D. Siegel, T. de Carvalho, and T. Nijsten, "Accuracy of a smartphone application for triage of skin lesions based on machine learning algorithms," *Journal of the European Academy of Dermatology and Venereology*, vol. 34, no. 3, pp. 648–655, 2020.

[7] K. C. Wong, H. Klimis, N. Lowres, A. von Huben, S. Marschner, and C. K. Chow, "Diagnostic accuracy of handheld electrocardiogram devices in detecting atrial fibrillation in adults in community versus hospital settings: a systematic review and meta-analysis," *Heart*, vol. 106, no. 16, pp. 1211–1217, 2020.

[8] R. C. Gibbons, D. J. Jaeger, M. Berger, M. Magee, C. Shaffer, and T. G. Costantino, "Diagnostic accuracy of a handheld ultrasound vs a cart-based model: A randomized clinical trial," *Western Journal of Emergency Medicine*, vol. 25, no. 2, p. 268, 2024.

[9] NVIDIA, "Jetson orin," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/, 2024, accessed: 2024-07-02.

[10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[11] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLO," Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37.

[14] M. Akay, Y. Du, C. L. Sershen, M. Wu, T. Y. Chen, S. Assassi, C. Mohan, and Y. M. Akay, "Deep learning classification of systemic sclerosis skin using the mobilenetv2 model," *IEEE Open Journal of Engineering in Medicine and Biology*, vol. 2, pp. 104–110, 2021.

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[16] H. Yu, C. Chen, X. Du, Y. Li, A. Rashwan, L. Hou, P. Jin, F. Yang, F. Liu, J. Kim, and J. Li, "TensorFlow Model Garden," https://github.com/tensorflow/models, 2020.

[17] D. Reis, J. Kupec, J. Hong, and A. Daoudi, "Real-time flying object detection with yolov8," *arXiv preprint arXiv:2305.09972*, 2023.

[18] U. C. Turba, R. Uflacker, C. Hannegan, and J. B. Selby, "Anatomic relationship of the internaljugular vein and the common carotid artery applied to percutaneous transjugular procedures," *CardioVascular and Interventional Radiology*, vol. 28, pp. 303–306, 2005.

[19] C. A. Troianos, R. J. Kuwik, J. R. Pasqual, A. J. Lim, and D. P. Odasso, "Internal jugular vein and carotid artery anatomic relation as determined by ultrasonography," *The Journal of the American Society of Anesthesiologists*, vol. 85, no. 1, pp. 43–48, 1996.

[20] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

[21] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1325–1334.

[22] Z. Towfic, D. Ogbe, J. Sauvageau, D. Sheldon, A. Jongeling, S. Chien, F. Mirza, E. Dunkel, J. Swope, M. Ogut *et al.*, "Benchmarking and testing of qualcomm snapdragon system-on-chip for jpl space applications and missions," in *2022 IEEE Aerospace Conference (AERO)*. IEEE, 2022, pp. 1–12.

[23] A. Sabater, L. Montesano, and A. C. Murillo, "Robust and efficient post-processing for video object detection," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 10 536–10 542.

[24] Y. Shi, N. Wang, and X. Guo, "Yolov: Making still image object detectors great at video object detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 2, 2023, pp. 2254–2262.

[25] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.