Quantitative Echocardiography: Real-Time Quality Estimation and View Classification Implemented on a Mobile Android Device

Nathan Van Woudenberg[1], Zhibin Liao[1], Amir H. Abdi[1], Hani Girgis[2], Christina Luong[2], Hooman Vaseli[1], Delaram Behnami[1], Haotian Zhang[1], Kenneth Gin[2], Robert Rohling[1], Teresa Tsang[2](✉), and Purang Abolmaesumi[1](✉)

[1] University of British Columbia, Vancouver, BC, Canada
purang@ece.ubc.ca
[2] Vancouver General Hospital, Vancouver, BC, Canada
t.tsang@ubc.ca

Abstract. Accurate diagnosis in cardiac ultrasound requires high quality images, containing different specific features and structures depending on which of the 14 standard cardiac views the operator is attempting to acquire. Inexperienced operators can have a great deal of difficulty recognizing these features and thus can fail to capture diagnostically relevant heart cines. This project aims to mitigate this challenge by providing operators with real-time feedback in the form of view classification and quality estimation. Our system uses a frame grabber to capture the raw video output of the ultrasound machine, which is then fed into an Android mobile device, running a customized mobile implementation of the TensorFlow inference engine. By multi-threading four TensorFlow instances together, we are able to run the system at 30 Hz with a latency of under 0.4 s.

Keywords: Echocardiography · Deep learning · Mobile · Real time

1 Introduction

Ischaemic heart disease is the primary cause of death worldwide. Practicing effective preventative medicine of cardiovascular disease requires an imaging modality that can produce diagnostically relevant images, while at the same time being widely available, non-invasive, and cost-effective. Currently, the method that best fits these requirements is cardiac ultrasound (echocardiography, echo). Modern echo probes can be used to quickly and effectively evaluate the health of the patient's heart by assessing its internal structure and function [3]. The major

T. Tsang and P. Abolmaesumi—Joint senior authors.

caveat of this process is that the interpretation of these images is highly subject to the overall image quality of the captured cines, which, in turn, is dependent on both the patient's anatomy and the operator's skill. Poor quality echoes captured by inexperienced operators can jeopardize clinician interpretation and can thus adversely impact patient outcomes [8]. With the proliferation of portable ultrasound technology, more and more inexperienced users are picking up ultrasound probes and attempting to capture diagnostically relevant cardiac echoes without the required experience, skill or knowledge of heart anatomy.

In addition to the task of acquiring high quality images, ultrasound operators can also be expected to acquire up to 14 different cross-sectional 'views' of the heart, each with their own set of signature features. Some of these views are quite similar to an inexperience eye, and switching between them can require very precise adjustments of the probe's position and orientation. In point-of-care ultrasound (POCUS) environments, the four views most frequently acquired by clinicians are apical four-chamber (AP4), parasternal long axis (PLAX), parasternal short axis at the papillary muscle level (PSAX-PM), and subcostal four-chamber (SUBC4).

In this work, we attempt to reduce the adverse effect of inter-operator variability on the quality of the acquired cardiac echoes acquired. The system we developed attempts to do this by providing the user with real-time feedback of both view classification and image quality. This is done through the use of a deep learning neural network, capable of simultaneous 14-class view classification and a quality estimation score. Furthermore, we implemented the system in

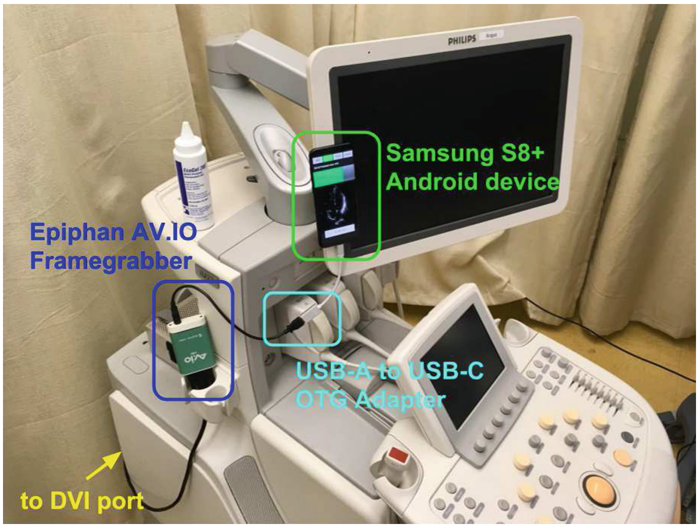

Fig. 1. The physical system setup. The frame grabber connects to the DVI output of the ultrasound machine. It is then connected to an OTG adapter and plugged directly into the Android's USB-C port.

the form of an Android application, and ran it on an off-the-shelf Samsung S8+ mobile phone, with the goal of making our system portable and cost effective. As shown in Fig. 1, the system receives its input directly from the DVI port of the ultrasound machine, using an Epiphan AV.IO frame grabber to capture and convert the raw video output to a serial data stream. The frame grabber output is then adapted from USB-A to USB-C with a standard On-The-Go (OTG) adapter, allowing us to pipe the ultrasound machine's video output directly into the Android device and through a neural network running on its CPU, using TensorFlow's Java inference interface. The classified view and its associated quality score are then displayed in the app's graphical user interface (GUI) as feedback to the operator. Figure 2 shows the feedback displayed in the GUI for four AP4 cines of differing quality levels. These four sample cines, from left to right, were scored by our expert echocardiographer as having image quality of 25%, 50%, 75%, and 100%, respectively.

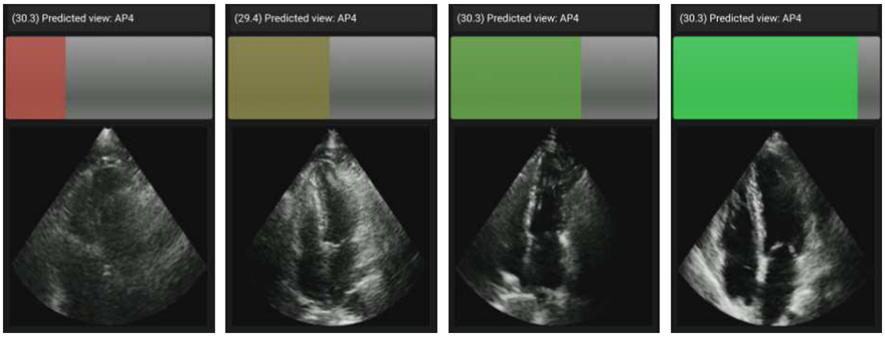
(30.3) Predicted view: AP4
(29.4) Predicted view: AP4
(30.3) Predicted view: AP4
(30.3) Predicted view: AP4

Fig. 2. The mobile application GUI showing the predicted view and quality for four different AP4 cines of increasing quality.

2 System Design

2.1 Deep Learning Design

A single deep learning network is used to learn the echo quality prediction and view classification for all 14 views. The model was trained on a dataset of over 16 K cines, distributed across the 14 views as shown in the following table:

Window	Apical				Parasternal						Subcostal			Suprasternal
view	AP2	AP3	AP4	AP5	PLAX	RVIF	PSAX_A	PSAX_M	PSAX_{PM}	PSAX_{AP}	SC4	SC5	IVC	SUPRA
# of cines	1,928	2,094	2,165	541	2,745	373	2,126	2,264	823	106	759	54	718	76

The network architecture can be seen in Fig. 3. The input to the network is a ten-frame tensor randomly extracted from an echo cine, and each frame is a

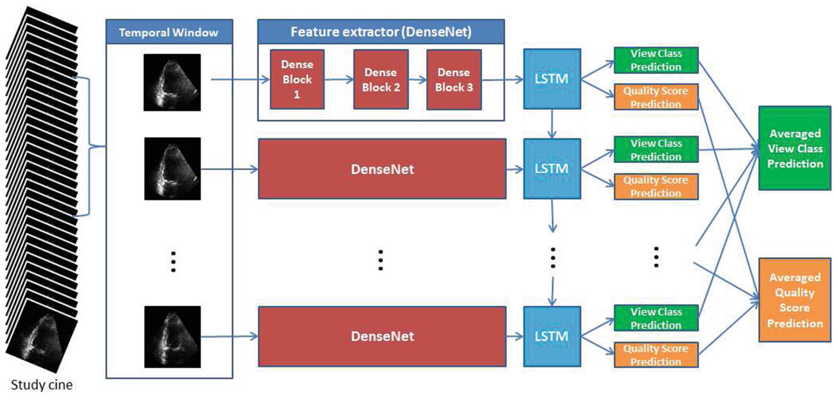

Fig. 3. The network architecture. Relevant features are extracted from the individual frames by the DenseNet blocks, which are then fed into the Long Short-Term Memory (LSTM) blocks to extract the temporal information across ten sequential echo cine frames.

120×120 pixel, gray-scale image. The network has four components, as shown in Fig. 3: (1) A seven-layer DenseNet [5] model that extracts per-frame features from the input; (2) an LSTM [4] layer with 128 units that captures the temporal dependencies from the generated DenseNet features, which produces another set of features, one for each frame; (3) a regression layer that produces the quality score from the output feature of the LSTM layer for each frame; and (4) a softmax classifier that predicts the content view from the LSTM features for each frame.

Our DenseNet model uses the following hyper-parameters. First, the DenseNet has one convolution layer with sixteen 3×3 filters, which turns the gray-scale (1-channel) input images to sixteen channels. Then, the DenseNet stacks three dense blocks, each followed by a dropout layer and an average-pooling layer with filter size of 2×2. Each dense block has exactly one dense-layer, which consists of a batch-normalized [6] convolution layer with six 3×3 filters and a Rectified Linear Unit (ReLU) [7] activation function. Finally, the per-frame quality scores and view predictions are averaged, respectively, to produce the final score and prediction for the ten-frame tensor.

2.2 Split Model

Initially, our system suffered from high latency due to the long inference times associated with running the entire network on an Android CPU. Since the network contains a ten-frame LSTM, we needed to buffer ten frames into a $120 \times 120 \times 10$ tensor, then run that tensor through both the Dense and LSTM layers of the network before getting any result. This produced a latency of up to 1.5 s, which users found frustrating and ultimately detrimental to the usefulness of the system.

In order to reduce the latency of the feedback, we split the previously described network into two sections: the Convolution Neural Network (CNN) section, which performs the feature extraction on each frame as they come in, and the Recurrent Neural Network (RNN) section, which runs on tensors now containing the features extracted from the previous ten frames. With the split model, we can essentially parallelize the feature extracting CNNs and the quality predicting RNN. See Fig. 6 for a visual view of the CNN/RNN timing.

2.3 Software Architecture

Figure 4 shows the data flow pipeline of the application. Input frames are captured by the frame grabber and are fed into the mobile application's Main Activity at a resolution of 640 × 480 at 30 Hz. We created a customized version of the UVCCamera library, openly licensed under Apache License, to access the frame grabber as an external web camera [2]. The application then crops the raw frames down to include only the ultrasound beam, the boundaries of which can be adjusted by the user. The cropped data is resized down to 120 × 120 to match the network's input dimensions. A copy of the full-resolution data is also saved for later expert evaluation. The resized data is then sent to an instance of TensorFlow Runner, a custom class responsible for preparing and running our data through the Android-Java implementation of the TensorFlow inference engine [1]. Here, we first perform a simple contrast enhancement step to mitigate the quality degradation introduced by the frame grabber. The frames are then sent to one of three identical Convolutional Neural Networks (CNN-1, CNN-2, or CNN-3). Each CNN runs in a separate thread in order to prevent lag during particularly long inference times. The extracted features are saved into a feature buffer which shared between all three threads. Once the shared feature buffer fills, the RNN thread is woken up and runs the buffered data through the LSTM portion of the network to produce the classification and quality predictions to be displayed in the GUI.

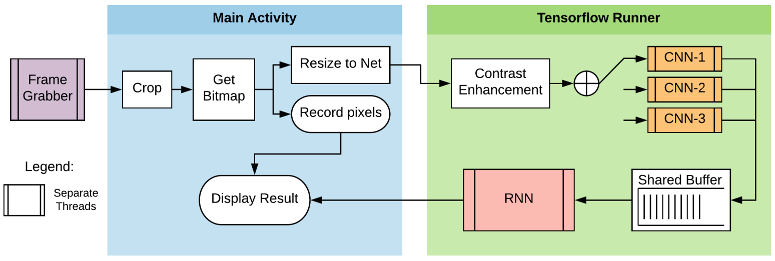

Fig. 4. Flow diagram of the software design.

3 Results

3.1 Classification

The training accuracy for the view classification was 92.35%, with a test accuracy of 86.21%. From the confusion matrix shown in Fig. 5, we can see that the majority of the classification error results from the parasternal short axis views, specifically PSAX$_M$, PSAX$_{PM}$, and PSAX$_{APIX}$. These 3 views are quite similar both visually and anatomically, and some of the cines in our training set contain frames from multiple PSAX views which may be confusing our classifier. The subcostal 5-chamber view also performed poorly, due to the small number of SC5 cines in our training set.

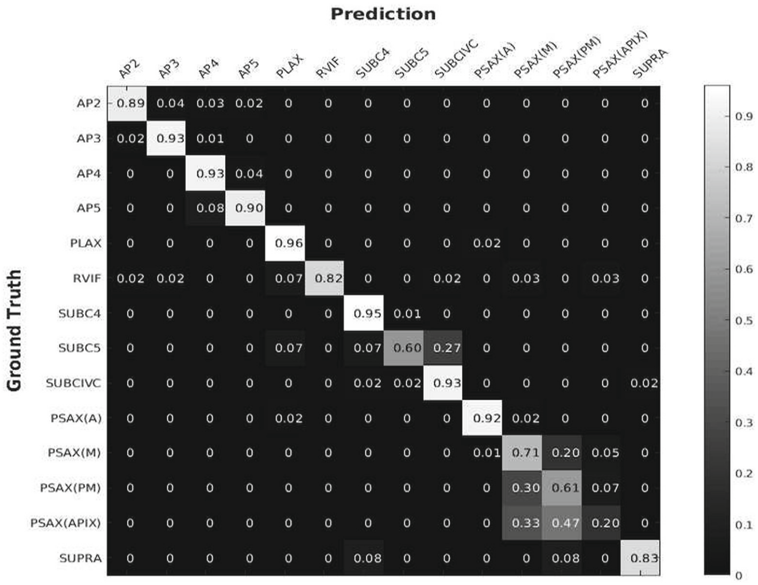

Fig. 5. The Confusion Matrix of the view classifier, showing all 14 heart views.

3.2 Timing

Since the system is required to run in real time on live data, the details regarding the timing are important to evaluating its performance. Figure 6 shows the timing profile of the three CNN threads, along with the single RNN thread, collected through Android Studio's CPU profiler tool. The three CNNs can be seen extracting features from ten consecutive input frames before waking the waiting RNN thread, which then runs the quality prediction on the buffered features extracted by the CNNs. The target frame rate for the system is set at 30 Hz, which can be inferred by the orange lines representing the arrival of

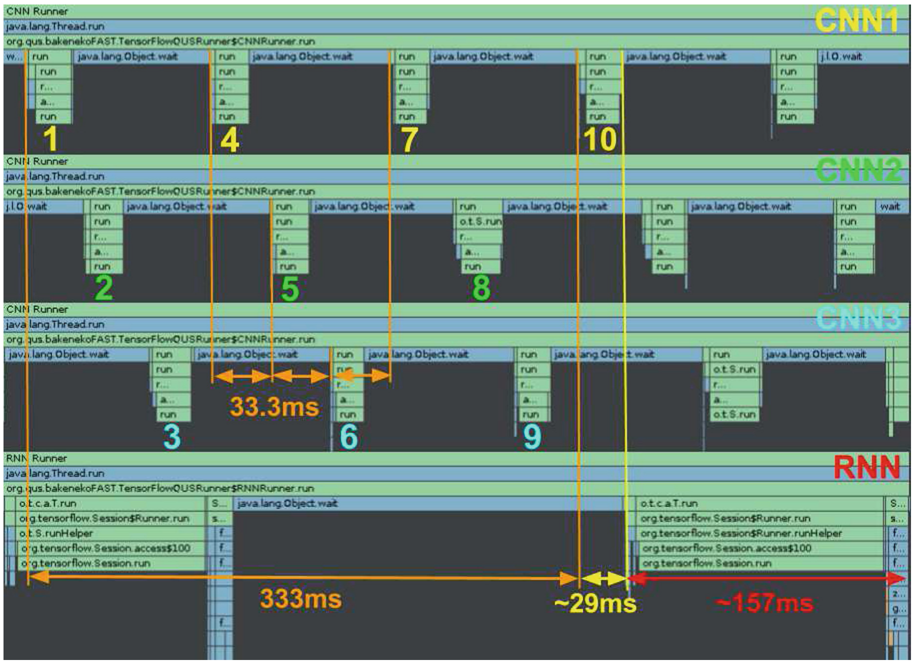

Fig. 6. Timing diagram of the three CNN and one RNN threads. The orange lines show the arrival of the input frames.

input frames. The mean CNN run-time (including feeding the input, running the network, and fetching the output) is 28.76 ms with an standard deviation of 16.42 ms. The mean run time of the RNN is 157.48 ms with a standard deviation of 21.85 ms. Therefore, the mean latency of the feedback is 352.58 ± 38.27 ms, when measured from the middle of the ten-frame sequence.

In order to prevent lag resulting from the build-up of unprocessed frames, the CNNs and RNN need to finish running before they are requested to process the next batch of data. To accomplish this reliably, all the per-frame processing must complete within $T_{max,CNN}$, calculated as follows:

$$T_{max,CNN} = (\text{\# of CNNs}) \times \frac{1}{\text{FPS}} = \frac{3}{30} = 100\ ms \tag{1}$$

while the RNN needs to complete its processing before the features from the next ten frames are extracted:

$$T_{max,RNN} = (\text{buffer length}) \times \frac{1}{\text{FPS}} = \frac{10}{30} = 333.33\,\text{ms} \tag{2}$$

With the chosen three-CNN-one-RNN configuration, the application required the fewest number of threads while still providing enough tolerance to avoid frame build-up.

4 Discussion

In this paper, we present a system that provides ultrasound operators with real-time feedback about the heart echoes being captured, in the form of view classification and image quality estimation. The system is implemented in an Android application on an off-the-shelf Samsung S8+ and can be connected to any ultrasound machine with a DVI output port. In order to reduce the latency of the system, the neural network is split into two sections: the CNN and the RNN, allowing us to parallelize their execution. With the split model, the system is able to operate at 30 frames per second, while providing feedback with a mean latency of 352.91 ± 38.27 ms.

The next step of this project is to validate the system in a clinical setting. Our group is currently running a study at Vancouver General Hospital, in which we ask subjects to acquire cines of the four POCUS views once with and once without displaying the quality and view feedback in the app. The two datasets will be scored by expert echocardiographers and then compared in order to quantify the accuracy and utility of the system. We also plan to migrate the backend to TensorFlow Lite, a lightweight implementation of the inference engine, which will allow us to leverage the hardware acceleration available on modern Android devices to help us further reduce the system's latency.

Acknowledgements. The authors wish to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canadian Institutes for Health Research (CIHR) for funding this project. We would like to also thank Dale Hawley from the Vancouver Coastal Health Information Technology for providing us access to the echo data during the development of this project.

References

1. Tensorflow android camera demo. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android. Accessed 4 Feb 2018
2. Uvccamera. https://github.com/saki4510t/UVCCamera. Accessed 16 Dec 2017
3. Ciampi, Q., Pratali, L., Citro, R., Piacenti, M., Villari, B., Picano, E.: Identification of responders to cardiac resynchronization therapy by contractile reserve during stress echocardiography. Eur. J. Heart Failure **11**(5), 489–496 (2009)
4. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
5. Huang, G., Liu, Z., Weinberger, K.Q., van der Maaten, L.: Densely connected convolutional networks. In: IEEE CVPR, vol. 1–2, p. 3 (2017)
6. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, pp. 448–456. JMLR (2015)
7. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-2010), pp. 807–814 (2010)
8. Tighe, D.A., et al.: Influence of image quality on the accuracy of real time three-dimensional echocardiography to measure left ventricular volumes in unselected patients: a comparison with gated-spect imaging. Echocardiography **24**(10), 1073–1080 (2007)