Democratizing Deep Learning-Based Non-Contact Vitals

Anand Sekar
University of Washington
Seattle, WA
anand272@cs.washington.edu

Xin Liu
University of Washington
Seattle, WA
xliu0@cs.washington.edu

Shwetak Patel
University of Washington
Seattle, WA
shwetak@cs.washington.edu

ABSTRACT

Non-contact, camera-based physiological measurements - such as blood volume pulse and respiration rate - can now be inferred by neural networks based on facial videos. This technology has the potential to enable medical professionals to make more informed telehealth decisions. Currently, this software only runs on PCs, without a user interface. The neural network has a significant computational cost, making it difficult to deploy on low-cost mobile devices. It also performs poorly in varied environmental, sensor, personal, and contextual conditions - such as darker skin tones. In this project, we implement this neural network as an Android app that runs in real-time; develop a more efficient architecture; evaluate these architectures on older smartphones; and provide an open-source, simple personalization pipeline to enable users to calibrate the app. This all serves to make the technology more democratic: making it available to as many users as possible, while giving them the means to train and develop it further.

Author Keywords

Non-contact vitals; Tensorflow lite; Android; photoplethysmography; PPG; multi-task temporal shift convolutional attention networks; telemedicine; R-PPG; ubiquitous computing

CCS Concepts

•**Human-centered computing** → *Smartphones;* •**Applied computing** → *Consumer health;*

INTRODUCTION

Along with the recent surge of cutting-edge information and communication technologies for development (ICT4D) rose a vast potential for healthcare applications. Digital health, also known by several other names such as telehealth or telemedicine, is largely composed of mobile, wireless health interventions. The ubiquity of mobile and wireless technologies, such as smartphones, makes them powerful tools for delivering medical care to users: distributing healthcare commodities across geographical barriers, making them more available and equitable. The World Health Organization (WHO) Recommendations on Digital Interventions for Health System Strengthening include increasing the availability of human resources for health with client-to-provider telemedicine:

"WHO recommends the use of client-to-provider telemedicine to complement, rather than replace, the delivery of health services and in settings where patient safety, privacy, traceability, accountability and security can be monitored" [35]. This includes the "remote monitoring of vital signs or diagnostic data." With respect to effectiveness, this may "improve some outcomes, such as fewer unnecessary clinical visits [and] reduced mortality among individuals with heart-related conditions" [35].

One vital sign commonly monitored is blood volume pulse (BVP) from a photoplethysmogram (PPG) signal. Another is respiration volume from a ballistocardiogram (BCG) signal. The SARS-CoV-2 (COVID-19) virus, for instance, has a host of cardiopulmonary symptoms; monitoring these signals can help physicians make more informed decisions. The COVID-19 pandemic has shown a drastic increase in telehealth usage, and has underscored not just the need for telehealth vital monitoring tools, but ones that are remote and contactless to reduce viral transmission.

Earlier this year, Liu et al. proposed a novel "multi-task temporal shift convolutional attention network (MTTS-CAN)" to address this problem [15]. It demonstrated state-of-the-art efficiency, and was relatively accurate with a mean absolute error (MAE) of 1.45. It was evaluated using the CPU of an embedded system as a proxy for mobile devices, but hadn't yet been implemented on consumer-grade hardware like smartphones. Live processing was only presumed to be feasible.

Additionally, it shared an issue with many other recent AI technologies: a lack of fairness. Deep learning-based facial-recognition software, for instance, is disproportionately inaccurate when used by women and people with darker skin color; women with darker skin color have error rates of up to 34.7% [7]. Likewise, the MTTS-CAN was trained on a couple, relatively small datasets. It's very sensitive to environmental (lighting), sensor (hardware), individual (skin type, facial hair), and contextual (what the person is doing) differences [16]. When just considering skin type, the MAE for darker skin types was approximately 9 times worse than the MAE for lighter skin types [16].

The other issue it shared was a lack of availability and top-down development. These cutting-edge neural networks aren't usually accessible to users on devices like smartphones. If they are, they don't perform well on older, cheaper devices.

CHI'20, April 25–30, 2020, Honolulu, HI, USA

ISBN 978-1-4503-6708-0/20/04…$15.00

DOI: https://doi.org/10.1145/3313831.XXXXXXX

Even then, they aren't open to being changed by the users themselves. ICT4D emphasizes the social good that results from interventions which focus on empowering users to help create technologies as opposed to prescribing solutions for them.

In this paper, we address the above issues respectively through the following contributions:

1. Implementing deep learning-based remote physiological sensing on an Android app called DeepTricorder.
2. Developing an efficient, single-branch network which reduces the computational load.
3. Evaluating the app on various architectures and lower-cost, lower-performance smartphones.
4. Democratizing the technology by enabling users to train the network themselves through an open-source personalization pipeline.

RELATED WORKS

Non-Contact Sensing

Heart-rate measurement and remote plethysmographic imaging began several years ago [29, 30] by analyzing how light interacts with the skin. This is also commonly called "rPPG" for remote photoplethysmography. These methods are based on optical models such as the Lambert-Beer Law and Shafer's dichromatic reflection model, which focus on slight changes in color under the skin from hemoglobin levels that fluctuate with pulse. Efforts to extract PPG signals using traditional, hand-crafted signal processing pipelines based off these models suffered from sensitivity to noise - such as head motions and illumination changes [22, 21]. Some approaches included prior information about the user's skin [11, 34]. In general, it was difficult to capture the spatio-temporal complexity of physiological signals hidden in a video. Neural networks have been successful in extracting the PPG signal [9, 28, 37, 27], but come with a high computational load that's too much for real-time performance.

Recent work on these neural networks emphasize improving performance with 3D convolutional neural network (CNN) architectures [37]. Although more accurate than their 2D counterparts, 3D CNNs require far more computation. Temporal shift modules are a clever mechanism which retain both the accuracy of 3D CNNs and the computational cost of 2D CNNs [36].

The Original Network

One of our goals is to build a computationally efficient on-device architecture that enables inference to occur real-time. We start with a state-of-the-art architecture, MTTS-CAN, for remote cardiopulmonary monitoring. MTTS-CAN is an on-device efficient neural architecture with two branches: an appearance branch and a motion branch. The input of the motion branch is the difference of every two consecutive frames and the output is the first-derivative of pulse. The input of the appearance branch is the raw video frames and and the outputs are two attention masks. The attention masks are used in the motion branch to help focus on regions of interest relevant to the physiological signal. In the motion branch, MTTS-CAN also leverages temporal shift modules (TSM) to exchange information across the time domain. Unlike 3D CNNs or long short-term memory (LSTM) components, TSMs capture temporal dependencies across consecutive frames with zero computational overhead. However, Liu et al. [15] only evaluated their network inference on an embedded system and didn't conduct a full end-to-end implementation (e.g. with preprocessing) or evaluation on actual smartphones. Therefore, the real-time inference with MTTS-CAN on mobile devices wasn't yet proven to work.

Ubiquitous Health Apps

This project is situated within the field of ubiquitous computing, specifically health sensing using common technology. Some examples include: Seismo, an app which implements blood pressure monitoring using build-in smartphone accelerometer and camera; HemaApp, an app which noninvasively screens blood hemoglobin using smartphone cameras; and SpiroCall, which measures lung function over a phone call [33, 32, 13].

There have been a few apps which measure vitals using the front-facing camera [6, 25]. There have also been a few apps which measure the PPG signal by using the back-facing camera, flashlight, and the index finger [26]. These target the same property fingertip pulse oximiters do, except they use reflectance instead of transmission. A recent feature added onto the Google Fit app does exactly this [20].

There have also been a whole generation of apps which use deep learning. One API which powers some of these applications is TensorFlow lite, which comes with several Android examples such as gesture recognition, object detection, and image classification [1]. Some of these apps are also for healthcare purposes. For instance, Contect is one such app that analyzes a patient's speech to detect possible neurological complications [5].

As far as we know, DeepTricorder is the first open-source, deep learning-based, non-contact vitals measurement mobile app.

Democratizing Technology

Recent ICT4D work re-conceptualizes technology development to empower the people it serves. Poveda & Roberts explore two ICT4D interventions to argue for a human-focused, critical-agency approach that emphasizes participants utilizing the technology "to act and bring about development changes they had [critically] reasoned and valued," to address the root, structural problems they face; this "contrasts with ICT4D initiatives in which ICT provision and skills are seen as ends in themselves or uncritically assumed to lead to economic development" [24]. Waldman-Brown et al. constructively criticizes the Maker Movement, contrasting "grassroots innovators" with "bourgeois hobbyists" who are privileged to already have access to a plethora of advanced, technological resources [31]. A critique of similar health interventions is that they tend to focus more on the development of the technology itself rather than the people and problems they address. A

critique of large deep learning networks is that although they are trained on data created by the public, they are not freely available to the public. Several theories have addressed knowledge, such as that which is used to train AI, as a common resource [8, 10]. Putting all this together, we contextualize our contributions within the scope of democratizing technology for social good, i.e. making not just the end product more available, but also the means.

METHODS

In this paper, we explore efficient on-device neural network architectures and aim for developing an end-to-end mobile system on smartphones at different price ranges. More specifically, we hope our system could run at real-time speed on mobile devices under $100 and help improve health equity.

Single-Branch Network

We propose a variant of MTTS-CAN to further reduce the inference time and cut the pre-processing time as well. Our solution is a single-branch neural network, as Figure 1 depicts. The purpose of the appearance branch is to generate two attention masks for the motion branch, so that the motion branch can focus on regions of interest containing the physiological signal. Therefore, we theorize that the motion branch could perform self-attention, removing the appearance branch entirely. As Figure 1 shows, we added two self-attention modules after the second and fourth convolutional operations. By doing so, we are able to reduce the computational load by almost half, but retain its ability to focus on region of interest. We hypothesize that our proposed network could save significant computational overhead on lower-end mobile devices and enable real-time non-contact physiological measurement.

End-to-End Implementation

Figure 2. Smartphones used for testing.

This app was created within the Android family of smartphones and apps. The app used the structure from TensorFlow Lite's image classification demo. Plotting the vital signals utilized the androidplot library [12]. The Butterworth filters in postprocessing were supported by Bernd Porr's IIRJ filter library [23].

The accuracy of the Android app with Java code was verified with respect to the original MTTS-CAN/ TS-CAN network as it runs on Windows with Python code. Results were compared at each stage of processing.

The app was developed on a Google Pixel 4A (2020). The app was subsequently tested on older smartphones: a Pixel 1 (2016), a Motorola Nexus 6 (2014), and a Motorola Moto G3 (2013) as shown in Figure 2. The Pixel4A costs $350, and since the rest aren't in production anymore, they were approximated to be $200, $150, and $100 respectively. The evaluation was performed by logging the time at each stage of processing over 10 iterations through Android's logcat debugging feature. Evaluating threading was performed similarly, but with 5 iterations averaged.

The personalization pipeline utilized a host of tools: a pulse sensor, an Arduino board, a webcam, the Windows 10 OS with batch command line executable files, the Anaconda data science platform for Python 3.8.8, Python libraries such as TensorFlow and OpenCV, and the Gradle build automation tool.

All the code is published as open-source in this Github repository: `https://github.com/ubicomplab/deep-rppg-android`.

RESULTS

App Overview

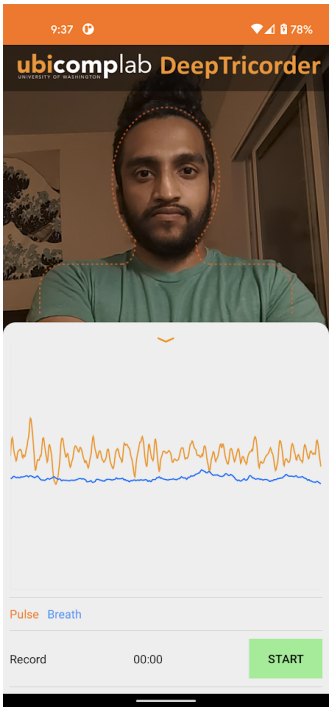

Figure 3. A screenshot of the app.

The app has a very simple user interface. As the user opens the app, it immediately begins calculating the pulse and breath signals. Figure 3 shows what the interface looks like. The user must place their face and shoulders within the dotted orange

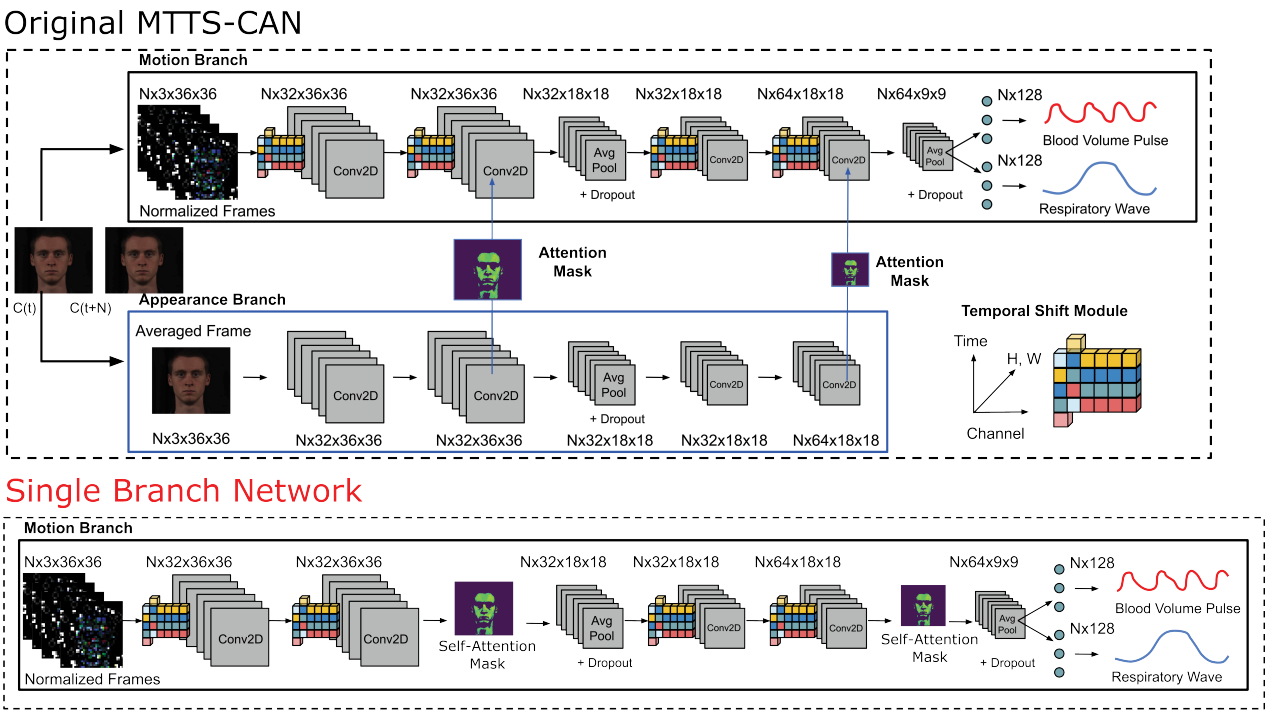

Figure 1. An illustration of the MTTS-CAN network architecture and our proposed single branch network architecture.

guideline. The orange line is the PPG/ pulse signal, and the blue line is the BCG/ breath signal. The bottom of the app includes a button which enables the user to record and save the values on the device.

For simplification, the analysis in this paper will refer to the TS-CAN network, i.e. the network that just outputs pulse; it essentially involves the same amount of computation as MTTS-CAN. Here are stages of processing - both on the app and on the full Windows/ Python code:

1. Image Capture. The app acquires a frame from the camera.

2. Crop. The app then immediately crops that frame into a smaller square which is big enough to capture the shoulders and face. These two steps are repeated until 20 frames have been collected for performing batch inference.

3. Resize. All the images are reduced into 36×36 pixel images. Since each pixel is represented by 3 values for red/ green/ blue (RGB), the data now takes the shape of $20 \times 36 \times 36 \times 3$ floating-point values.

4. Two copies of the data are made. One copy becomes the appearance branch, which normalizes each frame.

5. The other copy becomes the motion branch, which normalizes across adjacent frames.

6. Infer Vitals. The appearance and motion branches are packaged as inputs into the neural network, which outputs the derivative of pulse and breathing data. 20 values are outputted, corresponding to each frame.

7. Post-Process. These values are integrated over using a combination of cumulative summation and a running average. Then, noise is filtered out and the signal is smoothed through Butterworth bandpass filters.

Figure 4 shows how the main classes underlying the app interact with each other. CameraActivity.java can be considered as the "main" loop. It instantiates the APIs dealing with the camera, which includes disabling auto focusing and white balancing. It's necessary to disable these since the neural network is essentially detecting slight changes in color caused by hemoglobin levels under the skin; auto white balancing would interrupt that. The frames collected by the camera, which run at approximately 30 frames per second (fps), are displayed as the camera preview in the UI.

ClassifierActivity.java mainly serves to handle the OnImageAvailableListener, which is a software interrupt that fires whenever the camera has collected a frame. It also initializes the classifier. The processImage() method takes a frame, crops it, and saves it into a buffer. Once 20 frames have been collected, this activity spawns a new thread and runs inference while continuing to collect more frames for the next batch.

The reason why the inference thread is given slightly lower priority is that collecting frames is the longest aspect of inference by far, as shown in Figure 8. We want to ensure that the the main thread collecting frames is operating as quickly as possible. Even with the lowest priority, calling inference on a set of frames consistently finishes in the time it takes to collect 6 to 7 (out of 20) new frames on the Pixel 4A. The resulting values are sent back through ClassifierActivity to the CameraActivity to be displayed on the graph via a UI thread.

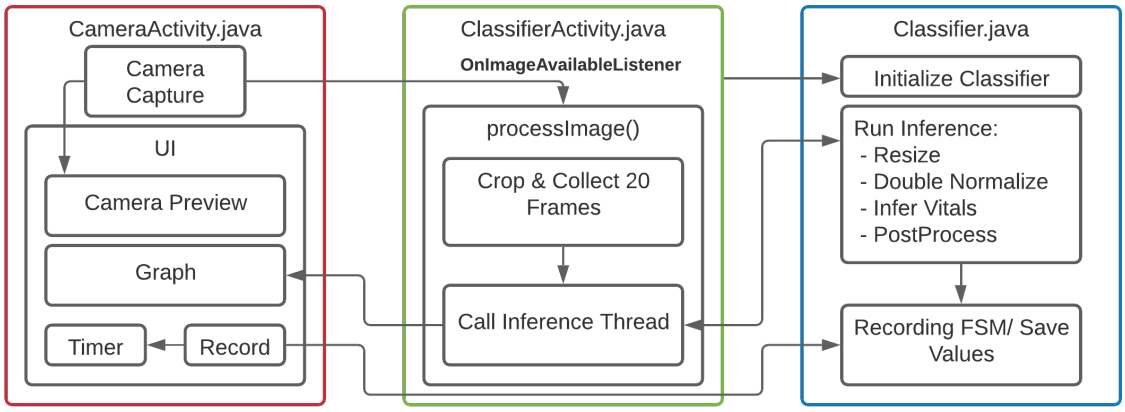

Figure 4. A system diagram documenting the structure of the app.

The record button toggles a boolean value visible to the Classifier, which keeps track of a finite state machine (FSM). When recording begins, a text file is created in the "Downloads" folder. When recording, the floating-point values outputted from the network (before post-processing) are rounded, separated by commas, and appended to the file. Once recording finishes, the resulting string is copied to the clipboard, and notifies the user of that.

App Verification

In order to verify that the app was performing the same computation as the original MTTS-CAN network, we compared each stage of processing side-by-side. The app's results - computed using TensorFlow Lite in Java on Android - will be referred to as "Lite" and the original results - computed using TensorFlow in Python on Windows - will be referred to as "Full".

The first stages of image capture to resizing (1-3) were relatively intuitive, and don't need side-by-side comparisons, as shown in Figure 5. They look similar to the way images on Full are captured, cropped, rotated, and resized. The rest of the verification process used 20 frames taken from a webcam video, resized using Full. This data, stored in 'resized.txt,' was fed into into processing steps 4 for both Full and Lite. From that point on, Full and Lite continued processing independently.

Both the appearance and motion branches calculated look identical when displayed side-by-side, as shown in Figure 5. The raw values underlying the visualization only differ in rounding on the order of $10^{-}15$, which is an insignificant difference.

The most important thing to verify was the final output of the networks, which is shown in Figure 6. The Full signal is shown in red and the Lite signal is shown in blue, both with some transparency. The outputs overlay each other nearly exactly - with the only difference being a one-digit precision difference at the $10^{-}7$ order of magnitude. The reason why the differences are at different orders of magnitude is that the earlier steps deal with double floating-point precision, whereas the final output is represented with single floating-point precision.

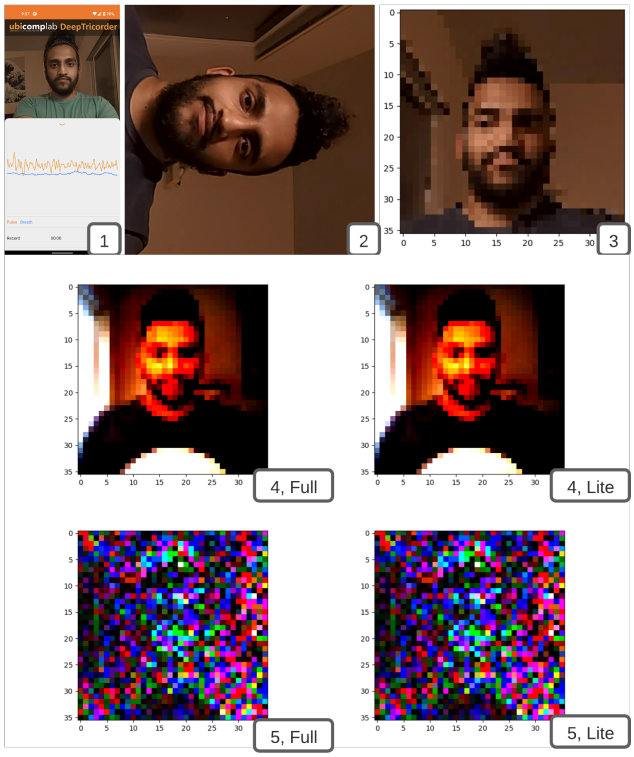

Figure 5. Preprocessing steps visualized.

The outputs after running inference are the values that are saved and can be shared, and we've verified that they match exactly with the original MTTS-CAN network. Post-processing, on the other hand, is done using a different method. Currently, post-processing on the app uses a cumulative sum and running average technique to deal with batches, whereas the original

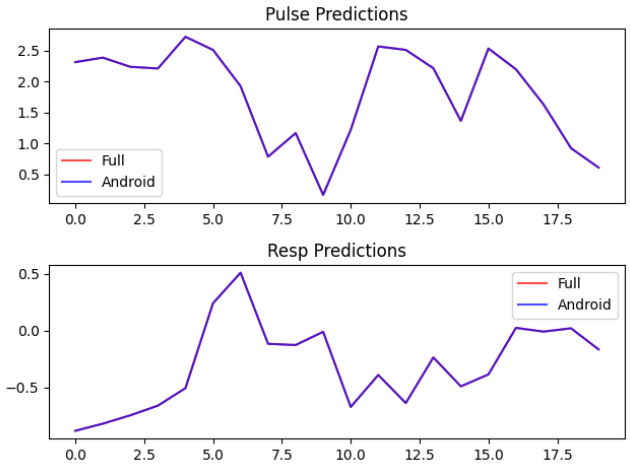

Figure 6. Output compared.

code is able to skip calculating a running average by performing de-trending. Both use Butterworth filters with the same parameters, but these most likely have different underlying implementations. The running average technique is derived from a web-based implementation of MTTS-CAN which is still under development. Since these post-processed values are only used to display to the user live, and post-processing is up to manual adjustment and development, an approximation served to be sufficient for the time being. Figure 7 shows how the Lite post-processing is similar enough to Full post-processing, with a MSE of approximately 0.15. The peaks and valleys correspond well.

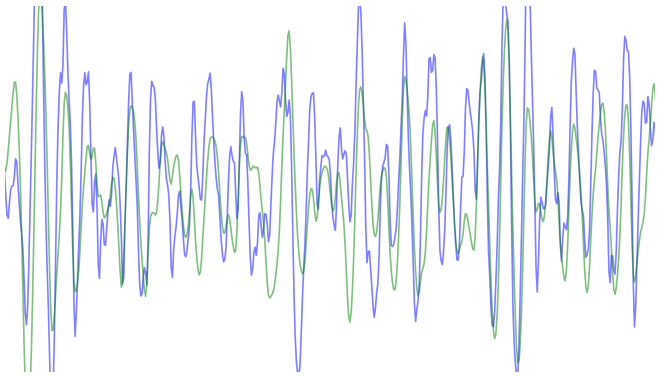

Figure 7. Post-processing compared. The blue line is Lite, and the green line is Full.

Evaluating Accuracy on Single-Branch Network

To evaluate the performance of our proposed network, we followed them same training scheme and configuration described in MTTS-CAN [15]. As Table 1 shows, the single-branch network only has a slight accuracy degradation compared with its original dual-branch version. This result reveals that the self-attention masks achieved similar effects as the attention masks generated from the appearance branch. However, the appearance branch almost doubled the computational time. With our newly-designed single-branch network, we were able to achieve strong performance with half of the computational time. To explore the effectiveness of the self-attention modules, we also implemented another single branch network without attention modules. As the Table 1 illustrates, without self-attention modules, the network has a 15% accuracy deficit, which indicates that self-attention modules play a significant role in training and inference.

Evaluating Efficiency on Different Devices

We evaluated the performance of the app - on different phones and different architectures - by printing timestamps before each step in the code, and averaging the results from 10 batches. Figure 8 shows a chart of what the processing times look like with TS-CAN on the Pixel 4A. The first two steps of capturing and cropping frames is done in parallel with all the other steps. The Pixel 4A has a camera which operates at approximately 30 fps, or $33.\bar{3}$ms per frame. Given that each batch is composed of 20 frames, and all computation is done in parallel, we would expect each batch to take $20 \times 33.\bar{3}\text{ms} = 666\text{ms}$; if you add up all the times in Figure 8, that's exactly the number we get. This also affirms why, as mentioned before, calling inference on a set of frames consistently finishes in the time it takes to collect 6 to 7 (out of 20) new frames: inferring vitals takes around a third of the time for each batch on the Pixel 4A.

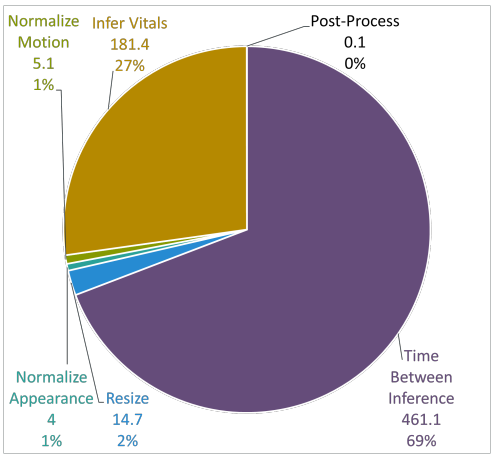

Figure 8. Processing times for each stage, as evaluated on the Pixel 4A. The label for each slice includes the processing stage, the time in milliseconds, and the percentage of the graph.

Table 2 summarizes the main comparisons. The "normalize appearance" step for the single-branch network is essentially zero because that step is skipped; it shows as non-zero due to time passed from threading. In every single case, except for normalizing motion, the single-branch architecture is significantly faster than the original architecture. This is most apparent when looking at the total inference time, and in the bottom row. The first couple Google smartphones, manufactured in 2020 and 2016 respectively, perform well and similarly; inferring vitals takes slightly longer on the Pixel 1, but still keeps up with live performance at 30 fps. The older

Method	MAE (Lower better)	SNR (Higher better)
MTTS-CAN	2.16	5.25
Single Branch Network	2.22	5.14
Single Branch without self-attention	2.47	4.23

Table 1. Accuracy comparison between MTTS-CAN and our proposed single branch network.

	Pixel 4A $350		Pixel 1 ~$200		Nexus 6 ~$150		Moto G3 ~$100	
Resize	14.7	13.1	31.8	26.8	58.3	55.3	78.5	65.2
Normalize Appearance	4	0	5.3	0.1	95.2	0.1	38.9	0.2
Normalize Motion	5.1	7.2	5.9	7.2	100	101.9	69.6	68
Infer Vitals	181.4	113.1	216.8	120.9	482.4	284.8	1280.3	692
Post-Process	0.1	0.2	0.1	0.3	0.5	0.3	0.3	0.2
Total Process	205.3	133.6	259.9	155.3	736.4	442.4	1467.6	825.6
Total Process Per Frame	10.265	6.68	12.995	7.765	36.82	22.12	73.38	41.28

Table 2. Comparing the performance of the original TS-CAN network (black) and our proposed, single-branch network (blue) on different phones. Times are in milliseconds.

phones, manufactured in 2014 and 2013 respectively, suffer in performance and cannot keep up real-time at 30 fps; the Moto 3 performs quite worse than the Nexus 6. However, the older phones benefit the most from the single-branch architecture, with an average 23.4ms less per frame, as opposed to just 4.4ms less per frame with the newer phones.

To see why this the older phones perform worse, we created a to-scale visual of how threading is occurring in these cases in Figure 9. Time is represented from left-to-right. The violet bar represents collecting and cropping frames, and the subsequent yellow bars represent processing and inference on those frames while the next batch of frames is being collected (represented by the next violet bar).

The bottom row shows how the Pixel phone is performing optimally: collecting and cropping frames at 30 fps, while finishing inference in a timely manner. This is essentially the case on the Pixel phones regardless of network architecture and version, lest slight differences. The width of the violet bars in this row represent live, optimal collection of 20 frames at 30fps, i.e. at 666 ms; a phone with more computational power, or a network architecture with less computational load, wouldn't lessen the width of this bar.

However, the width of the violet bars on the rest of the rows for other phones are significantly longer. The Nexus 6 is generally faster than the M3. The important thing to note is that the single-branch network reduces the inference time, and also ends up having a shorter time collecting frames. Despite that, even when no inference is being performed in parallel, the collection of frames is still significantly slower. Therefore, the primary bottleneck for these older phones isn't the neural network's computational load, it's the acquisition, cropping, and allocation of frames.

Increasing the thread priority would not solve this issue, since we're past the threshold of threading congestion. Before increasing the priority of the inference thread, i.e. when it was at minimum priority, inference wouldn't even begin until several batches had been collected, essentially resulting in multiple threads being spawned and computational congestion. Increasing the thread priority to medium enabled computation to finish before the next batch of frames were collected, avoiding that congestion, and producing the proper threading seen in Figure 9.

Moving the resizing step to occur before storing data into the batch buffer may help; however, the resizing itself takes significant computation, and the amount of computation associated with simply moving around the bigger, cropped image is not known. We suspect that Android's current camera API doesn't optimally capture the frames into bitmaps on the older smartphones. However, we also suspect that this could be solved by diving deeper into the API, since the camera preview displays smoothly on the UI.

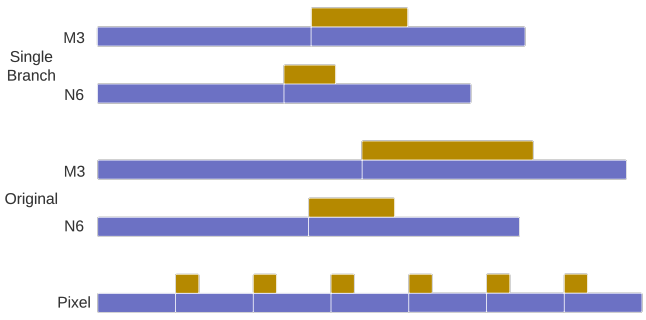

Figure 9. How the app threads on different architectures and phones; chronologically represented from left-to-right. "Original" represents the original TS-CAN network, "Single" represents our proposed, single-branch network, "M3" represents the Moto 3, and "N6" represents the Nexus 6. The violet bars represent collecting and cropping frames, and the yellow bars represent processing. The purple bars on the bottom are 666ms, and the rest are relatively scaled.

All this being said, the app is still usable even in the earlier phones: the data calculated, displayed, and recorded is the same, but just delayed. The recording feature takes a noticeably few more seconds to save values. The only disadvantage

users would have using these older phones, then, is not having the values displayed live, and taking longer to collect the data. However, with the delay being only 2-4 times as long, and assuming that only a minute's worth of data would be useful for a healthcare provider, the users with an old phone can still feasibly save and send the same values that would be calculated with the app running on a newer phone; this would make no difference to the healthcare provider.

Personalization Pipeline

The goal of the personalization pipeline is to enable users to customize the app to their conditions, giving them the means to train the network themselves. In order to train a neural network, one needs input data, output data, and a computer. The current personalization pipeline requires users to have a webcam, a pulse sensor, and a Windows computer. Although this allows developer-oriented users to train, it doesn't include those who aren't experienced with computers, or those who just have their smartphone. However, in the near future, this can all plausibly be done on the smartphone: the front-facing camera, the back-facing camera, and the smartphone's CPU (or GPU) itself. The back-facing camera can record the user's ground-truth PPG signal by having them place their finger over the back camera; this was demonstrated by the most recent Google Fit update for the Pixel model phones [20]. Once that data is collected and synchronized with video from the front-facing camera, the TensorFlow Lite library enables training on-device. For instance, the gesture recognition demo [4] enables user-friendly model training on a web app.

Furthermore, this framework opens up the possibility for federated learning: training the overall non-contact vitals network across users' phones globally, improving the robustness and availability of the network for everyone [3]. This results in distributing the means of neural network training to the masses, allowing the users to improve the app for themselves; this is in contrast with recent, cutting-edge AI products which use training data from the masses, but keep the means of development for themselves, distributing the product in a top-down manner.

All the user essentially has to do is ensure their webcam is working, get their pulse sensor's port number, ensure the pulse sensor is working (with help from a batch script that plots the data), plug in their Android phone, and finally double-click on the 'personalize.bat' file which will take care of the rest. This batch file:

1. Collects the user's data while displaying a live webcam preview.
2. Generates synchronized video frame and pulse data files.
3. Trains the existing model and updates weights with the new data.
4. Converts the new model into a .tflite file.
5. Replaces the .tflife file within the Android project.
6. Builds the updated app to produce a new Android package (APK).
7. Installs the updated APK onto the phone.

There are also batch files which reset and re-install the original network, in case the user wants to start over. Again, the usability and accessibility of this pipeline lacks relative to having a user simply click a button on a mobile app. However, it does make re-training and personalizing the network as approachable as installing new software on a PC. This makes it accessible for underserved users - who may have darker skin tones or different lighting conditions - to further develop the app for themselves and others. This could be the first step in a federated learning process where users around the world calibrate and update the app using contact-sensing, such that the non-contact inference becomes more robust and serves people more fairly.

From some initial testing, the personalized signal seems qualitatively smoother and less noise. It will require further evaluation in the near future. There are also other methods, such as few-shot adaptation and meta-learning [16], for calibrating the network in a more effective manner than additional training. Again, this personalization pipeline is accessible at https://github.com/ubicomplab/deep-rppg-android.

DISCUSSION

Solution Feasibility

The app works live on the newer phones, and slower-than-live on significantly older, cheaper phones. The single-branch network trades off some accuracy to minimize the computational impact, but there's still some work to be done to make it live. Most importantly, the app calculates the same values as those developed on computers using Python and libraries such as TensorFlow; regardless of whether or not the values can be displayed live on the older and cheaper phones, they will perform the same computation and net the same values. These values are copied to the clipboard and can be sent through any text medium to physicians for analysis. Since copying values to the clipboard can be a security vulnerability, future implementations of this app can use Android intents to securely share the data with other telemedicine apps. During the development process, we've made it relatively easy to test out different network architectures, and have made the training process as accessible to users with the current technology available. It implements recent research and puts it in the hands of users.

An important question must be answered: even if we implement the neural network on the app perfectly, and have trained the neural network rigorously on various lighting, skin, and situational contexts, will that kind of accurate signal still be useful to physicians? Throughout this research, we've primarily focused on getting the technology to be accurate with the datasets we have, and usable with technology we have. We are nowhere near ready for user studies or evaluation. While there is a whole field of ubiquitous health sensing which proves to be useful, this doesn't necessarily prove that this project within this domain would be useful.

Deep learning networks have recently proven to outperform more conventional methods, especially in complex tasks; but there are cases where simpler, conventional methods are most effective. For instance, a PID controller with a few lines of code may be better at balancing an inverted pendulum or

guiding a 2D navigation problem than a neural network that's well-trained [18, 17]. This would be analogous to the case with contact-sensing: using direct reflectance, as the new Google Fit app or raw pulse sensor do, with some signal processing would net a direct, proper, near-gold-truth signal; whereas using a neural network to perform inference on the reflectance input wouldn't be as true. That neural network may still perform accurately, but wouldn't beat the straightfoward, direct implementation. All this doesn't apply to this project, because non-contact sensing is inherently a complex task. There isn't a proper, gold-truth sensor that does more "direct" non-contact sensing. There are non-contact sensing apps out there that don't use deep learning to acquire the signal; there is still approximation associated with getting those values. In short, there isn't a medical-grade non-contact vitals sensor.

That being said, there is an critical caveat: in order to diagnose cardiac issues like arrhythmia, the signal needs to be represented in fine-detail. For instance, electrocardiogram (ECG) signals (a different, more precise measurement of heart rate) need to be detailed for analysis. One study concluded that "down-sampling to 50 Hz proved to be unacceptable for both time- and frequency-domain analyses. At 50 Hz, the root-mean-squared successive differences and the power of high frequency tended to have high values and random errors" [14]. However, we are not attempting to capture ECG, we are attempting to capture PPG, which can still viably detect cardiac arrhythmia [19]. Nevertheless, chances are we still need a similar rate of sampling. With a 30 frame-per-second sampling rate, how would the neural network be able to capture higher-frequency phenomena in the PPG signal? Moreover, the neural network would need to be trained on what an arrhythmic pulse looks like; otherwise, it may just be inferring what a smoother pulse should look like at a similar rate. Liu et al. [15] demonstrated that it performed more efficiently and accurately than non-deep-learning methods; but this only applies to the clean, controlled, non-diverse dataset. There needs to be further analysis on whether the neural network truly measures the signal, or just does a really good job at predicting and imitating it using the same information. This is somewhat reminiscent of how the app seemed optimally performant until we tested it on older phones and had to change the threading mechanics: we won't know unless we try it out and see what happens.

Contribution to Social Good

We presume that increasing availability to this technology also serves to increase equity. We've done so in four ways:

1. Made the neural network work on a user-friendly app.
2. Created a low-computational-load neural network that works better on low-cost devices.
3. Enabled users to relatively easily train the network themselves.
4. Made all this open-source.

The most justifiable aspect of this project is that it is open-source, can be trained by users, and has been made essentially democratic. The common theme with cutting-edge research projects like these is that even if the source code is open and well-documented, it isn't necessarily approachable. We haven't done any user-based testing, and our perspectives as computer scientists adds significant bias, but we believe that the personalization pipeline can be used and molded by tech-oriented users. Again, the future direction for this pipeline is to become part of the app, and be incorporated into federated learning accessible to all users. Creating a faster, more efficient network also directly increases the availability of the technology. Once some rough edges surrounding the camera API on the low-cost phones is smoothed out, we've already developed a low-computational-load network that will fit right in with the rest of the app. Having all this be open-source enables developers like us to take this project further, or at least serve as a well-documented example of recent APIs which aren't excellently documented. The previous question regarding feasibility still applies: do we know that this is something physicians could use in telemedicine? Our answer is that this technology is still budding - ripe for exploration. We won't know the answer until we let the project mature to true usability. All we can do is widen the opportunities for its growth by making it accessible for use, training, and development.

This also comes with a vast space of ethical considerations and trade-offs. For instance, the reason that OpenAI keeps GPT-3 as a commercial product hidden behind an API, despite being trained on public data, is to prevent harmful uses of the model, such as for disinformation [2]. They control who has access, and can ban malicious users. However, this places all the power on OpenAI: they must define the criteria of what applications are harmful. Since they depend on funding from the API as a commercial product, the average user has no say in that criteria. They are performing research into potential misuses and model biases, but that still doesn't empower users to eliminate bias. This is typically authoritarian, and although technology companies such as OpenAI can work consistently within that framework, we believe in something more democratic, i.e. viewing this software as a commons. There are malicious uses of this technology - people may be able to collect others' vitals without their whole consent, such as during interviews or interrogations. Keeping that technology to ourselves doesn't necessarily prevent this - researchers may receive government and police funding to implement this themselves. We believe that opening up this software, while enabling malicious users, also enables benevolent users and other researchers to detect, prevent, and mitigate future harmful applications during development.

CONCLUSION

Throughout this project, we've created an Android app that utilizes a neural network to measure non-contact vitals, developed an efficient architecture, tested it on low-cost smartphones, and provided an open-source personalization pipeline to empower users to customize the app. It's shown to be accurate, efficient, usable, and available. We conclude that at this early stage, the true accuracy of the app in a healthcare setting is yet to be proven; we also conclude that it can only be proven, or disproven, by developing further, and that this is best done alongside the diverse population of users who can now help it help themselves.

ACKNOWLEDGMENTS

We thank the staff of and our fellow classmates in CSE 599 H Sp21: Computing for Social Good for helping us contextualize this research. We thank the members of the Ubiquitous Computing Lab at the University of Washington for their support and feedback. Most of all, we thank our family and friends for wholly supporting us through the research process.

REFERENCES

[1] 2017. TensorFlow Lite | ML for Mobile and Edge Devices. (2017). https://www.tensorflow.org/lite

[2] 2020. OpenAI API. (June 2020). https://openai.com/blog/openai-api/

[3] 2021a. Federated learning. (May 2021). https://en.wikipedia.org/w/index.php?title=Federated_learning&oldid=1026078958 Page Version ID: 1026078958.

[4] 2021b. tensorflow/examples. (June 2021). https://github.com/tensorflow/examples/blob/24aef3acf2abe298c0111165e65d98ce845af467/lite/examples/gesture_classification/web/README.md original-date: 2018-07-16T22:11:56Z.

[5] Jose I. Benedetto, Pablo Sanabria, Andres Neyem, Jaime Navon, Christian Poellabauer, and Bryan (Ning) Xia. 2018. Deep Neural Networks on Mobile Healthcare Applications: Practical Recommendations. *Proceedings* 2, 19 (Oct. 2018), 550. DOI: http://dx.doi.org/10.3390/proceedings2190550

[6] Simone Benedetto, Christian Caldato, Darren C. Greenwood, Nicola Bartoli, Virginia Pensabene, and Paolo Actis. 2019. Remote heart rate monitoring - Assessment of the Facereader rPPg by Noldus. *PLoS ONE* 14, 11 (Nov. 2019). DOI: http://dx.doi.org/10.1371/journal.pone.0225592

[7] Joy Buolamwini and Timnit Gebru. 2018. Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. (2018), 15.

[8] Charlotte Hess and Elinor Ostromg. 2006. *Understanding Knowledge as a Commons | The MIT Press*. https://mitpress.mit.edu/books/understanding-knowledge-commons

[9] Weixuan Chen and Daniel McDuff. 2018. Deepphys: Video-based physiological measurement using convolutional attention networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 349–365.

[10] Kate Crawford and Vladan Joler. 2018. Anatomy of an AI System. (2018). https://anatomyof.ai/

[11] Gerard De Haan and Vincent Jeanne. 2013. Robust pulse rate from chrominance-based rPPG. *IEEE Transactions on Biomedical Engineering* 60, 10 (2013), 2878–2886.

[12] Nick Fellows. 2015. Androidplot. (2015). http://androidplot.com/

[13] Mayank Goel, Elliot Saba, Maia Stiber, Eric Whitmire, Josh Fromm, Eric C. Larson, Gaetano Borriello, and Shwetak N. Patel. 2016. SpiroCall: Measuring Lung Function over a Phone Call. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, San Jose California USA, 5675–5685. DOI:http://dx.doi.org/10.1145/2858036.2858401

[14] Ohhwan Kwon, Jinwoo Jeong, Hyung Bin Kim, In Ho Kwon, Song Yi Park, Ji Eun Kim, and Yuri Choi. 2018. Electrocardiogram Sampling Frequency Range Acceptable for Heart Rate Variability Analysis. *Healthcare Informatics Research* 24, 3 (July 2018), 198–206. DOI: http://dx.doi.org/10.4258/hir.2018.24.3.198

[15] Xin Liu, Josh Fromm, Shwetak Patel, and Daniel McDuff. 2020. Multi-Task Temporal Shift Attention Networks for On-Device Contactless Vitals Measurement. (2020), 12.

[16] Xin Liu, Ziheng Jiang, Josh Fromm, Xuhai Xu, Shwetak Patel, and Daniel McDuff. 2021. MetaPhys: few-shot adaptation for non-contact physiological measurement. In *Proceedings of the Conference on Health, Inference, and Learning*. ACM, Virtual Event USA, 154–163. DOI: http://dx.doi.org/10.1145/3450439.3451870

[17] Girish Narayanaswamy and Anand Sekar. 2021. *Get In The Gym! Solving OpenAI Gym Environments with RL*. Technical Report. https://drive.google.com/file/d/1BUbN03e0OCJtD_ZxHcCPkuQZUSDlCdzw/view?usp=sharing

[18] OpenAI. 2016. Gym: A toolkit for developing and comparing reinforcement learning algorithms. (2016). https://gym.openai.com

[19] Neeraj Paradkar and Shubhajit Roy Chowdhury. 2017. Cardiac arrhythmia detection using photoplethysmography. *Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual International Conference* 2017 (July 2017), 113–116. DOI: http://dx.doi.org/10.1109/EMBC.2017.8036775

[20] Shwetak N. Patel. 2021. Take a pulse on health and wellness with your phone. (Feb. 2021). https://blog.google/technology/health/take-pulse-health-and-wellness-your-phone/

[21] Ming-Zher Poh, Daniel J McDuff, and Rosalind W Picard. 2010a. Advancements in noncontact, multiparameter physiological measurements using a webcam. *IEEE transactions on biomedical engineering* 58, 1 (2010), 7–11.

[22] Ming-Zher Poh, Daniel J McDuff, and Rosalind W Picard. 2010b. Non-contact, automated cardiac pulse measurements using video imaging and blind source separation. *Optics express* 18, 10 (2010), 10762–10774.

[23] Bernd Porr. 2021. berndporr/iirj. (May 2021). https://github.com/berndporr/iirj original-date: 2016-10-07T22:07:59Z.

[24] Sammia Poveda and Tony Roberts. 2018. Critical agency and development: applying Freire and Sen to ICT4D in Zambia and Brazil. *Information Technology for Development* 24, 1 (Jan. 2018), 119–137. DOI: http://dx.doi.org/10.1080/02681102.2017.1328656 Publisher: Routledge _eprint: https://doi.org/10.1080/02681102.2017.1328656.

[25] Philipp Rouast. 2021. prouast/heartbeat-android. (May 2021). https://github.com/prouast/heartbeat-android original-date: 2016-08-01T18:58:18Z.

[26] Anand Samajdar. 2021. AnandS09/PPG_AndroidVersion. (May 2021). https://github.com/AnandS09/PPG_AndroidVersion original-date: 2016-02-11T18:25:07Z.

[27] Rencheng Song, Senle Zhang, Chang Li, Yunfei Zhang, Juan Cheng, and Xun Chen. 2020. Heart Rate Estimation from Facial Videos Using a Spatiotemporal Representation with Convolutional Neural Networks. *IEEE Transactions on Instrumentation and Measurement* (2020).

[28] Radim Špetlík, Vojtech Franc, and Jirí Matas. 2018. Visual heart rate estimation with convolutional neural network. In *Proceedings of the British Machine Vision Conference, Newcastle, UK*. 3–6.

[29] Chihiro Takano and Yuji Ohta. 2007. Heart rate measurement based on a time-lapse image. *Medical engineering & physics* 29, 8 (2007), 853–857.

[30] Wim Verkruysse, Lars O Svaasand, and J Stuart Nelson. 2008. Remote plethysmographic imaging using ambient light. *Optics express* 16, 26 (2008), 21434–21445.

[31] Anna Waldman-Brown, Juliet Wanyiri, Simeon Adebola, Tim Chege, and Marian Muthui. 2015. DEMOCRATISING TECHNOLOGY: THE CONFLUENCE OF MAKERS AND GRASSROOT INNOVATORS.

[32] Edward Jay Wang, William Li, Doug Hawkins, Terry Gernsheimer, Colette Norby-Slycord, and Shwetak N. Patel. 2016b. HemaApp: noninvasive blood screening of hemoglobin using smartphone cameras. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, Heidelberg Germany, 593–604. DOI: http://dx.doi.org/10.1145/2971648.2971653

[33] Edward Jay Wang, Junyi Zhu, Mohit Jain, Tien-Jui Lee, Elliot Saba, Lama Nachman, and Shwetak N. Patel. 2018. Seismo: Blood Pressure Monitoring using Built-in Smartphone Accelerometer and Camera. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, Montreal QC Canada, 1–9. DOI:http://dx.doi.org/10.1145/3173574.3173999

[34] Wenjin Wang, Albertus C den Brinker, Sander Stuijk, and Gerard de Haan. 2016a. Algorithmic principles of remote PPG. *IEEE Transactions on Biomedical Engineering* 64, 7 (2016), 1479–1491.

[35] World Health Organization. 2019. *WHO guideline.* http://www.ncbi.nlm.nih.gov/books/NBK541902/ OCLC: 1126668274.

[36] Tete Xiao, Quanfu Fan, Dan Gutfreund, Mathew Monfort, Aude Oliva, and Bolei Zhou. 2019. Reasoning About Human-Object Interactions Through Dual Attention Networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 3919–3928.

[37] Zitong Yu, Xiaobai Li, and Guoying Zhao. 2019. Remote photoplethysmograph signal measurement from facial videos using spatio-temporal networks. In *Proc. BMVC*. 1–12.