Cardiac anomaly detection models for wearable devices

Sudden Cardiac Arrest (SCA) is a devastating heart abnormality which leads to millions of casualties per year. Thus, early detection or prediction of SCA could save human lives in a greater scale. In this work, we aimed to predict SCA before its occurrence and significant results has been obtained using the proposed signal processing methodology. Taking these models to an embedded device is the main purpose of this work.

Dataset:

We chose the **MIT-BIH Arrhythmia Database**. An Arrhythmia dataset is enough to check for irregularities in a person's heartbeat. Since we are not classifying the beats with respect to its annotations, we did not check which abnormality occurs. The MIT-BIH dataset contains Electrocardiogram waveform data. We used the wfdb package to read the data from Physionet Database. It contains a total of 48 half-hour excerpts of two-channel ambulatory ECG recordings.

Models:

First, we take a look into the Physiobank Beat an-notations and classify them as normal and abnormal beats. We then extracted the label information using the specified package(wfdb) and split the hour-long excerpts into smaller chunks. The splitting of ECG data is done in order to get more training data. The dataset contains approximately 82,873 inputs after pre-processing.

To detect these abnormalities, we came up with 3 solutions:

- CNN model
- CNN+LSTM model
- Forest Classifier

We chose to use Random Forest as it is often used for time series forecasting. Since it is an ensemble method, it provides a good accuracy and handles outliers as well. A random forest handles these outliers by essentially binning them. As CNNs feature parameter sharing and dimensionality reduction, we chose them for the experiment. In addition to the CNN, we added an LSTM layer since it is capable of learning long-term dependencies.

1. **Convolutional neural network**

We made a lightweight two layered convolutional neural network. The model inputs the ECG data and outputs the beat type (normal or abnormal). The network consists of a convolutional layer and a Dense layer including a flatten layer and dropout of 0.5.

The accuracy of this model was approximately 96.19% with a validation accuracy of 81.43%.

FLOPS is a measure of computational performance of processors. Using TensorFlow libraries we calculated the Floating Operations Per Second (FLOPS) of our model, it had 17,315 FLOPS. A model of this computational capacity can be run on a micro-controller.

Details of the CNN model is as follows,

Current CNN model:
Accuracy-0.9619 Val_Accuracy-0.8143
Quantized model:
Accuracy-0.9614 Val_Accuracy-0.8642
FLOPS - 17315

Size:
The estimated model size is the number of parameters multiplied by the size of the data type. Here we used Float32, so we have to multiply by 4 bytes.

Note: We did not tweak with the parameters here so that can be improved and thus resulting in a smaller model.

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 2160, 8)	40
flatten (Flatten)	(None, 17280)	0
dropout (Dropout)	(None, 17280)	0
output (Dense)	(None, 1)	17281

Total params: 17,321
Trainable params: 17,321
Non-trainable params: 0

Size if the layers are,
Conv1D: 40 x 4 = 160 Bytes
Dense: 17281 x 4 = 69,124 bytes = 67.5 KB

Total size of CNN model = 67.504 KB
After Quantization (model becomes 1/4th the size): 16.87 KB

2. **CNN + LSTM model**

LSTMs can handle sequential data more efficiently than CNNs, which are small and efficient, but not as efficient during processing.
Our results could be confirmed by using a combination of convolutional neural network and a recurrent neural network architecture called Long Short-Term Memory (LSTM) to determine whether they corresponded to our data more closely.

We used one convolutional layer, a dense layer and a LSTM layer of 2 inputs in a TimeDistributed architecture to obtain our model. The accuracy of LSTM model was approximately 98.49% with a validation accuracy of 80.81%. We had around 51 thousand parameters with 86,621 FLOPS.

Details of the CNN+LSTM model is as follows,
The estimated model size is the number of parameters multiplied by the size of the data type. Here we used Float32, so we have to multiply by 4 bytes.

Note: We tweaked the model hyperparameters so as to get the smallest and most efficient model.

Layer (type)	Output Shape	Param #
time_distributed_18	(TimeDis (None, 4, 540, 4)	16
time_distributed_19	(TimeDis (None, 4, 2160)	0
lstm_9 (LSTM)	(None, 2)	17304
output (Dense)	(None, 1)	3

Total params: 17,323
Trainable params: 17,323
Non-trainable params: 0

time_distributed: 16 x 4 = 64 bytes
time_distributed_1: 0 bytes
LSTM: 17304 x 4 = 69,216 bytes = 67.59 KB
Dense: 3 x 4 = 12 bytes
Accuracy of the model = approximately 96.85%

Total size of CNN+LSTM model = 67.67 KB
After Quantization (model becomes 1/4th the size): 16.92 KB (FINAL SIZE)

3. **Random Forest Classifier**

Deep learning approaches are not always necessary. While they are more accurate on time series data, they are quite large in size compared to small machine learning algorithms. One such example is a Random Forest Classifier.

To apply Random Forests algorithm on time series data, we transform it into a supervised learning problem. To do so, we do the pre-processing mentioned in previous section, but instead of feeding these inputs into a neural network, we input these values into a csv file.

The random forest classifier was then implemented on it. We achieved a cross-validation accuracy of 85.8%.

TABLE I
ACCURACY SCORES OF METHODS ON THE MIT-BIT ARRHYTHMIA DATASET

Architecture	Acc. [%]	Val. Acc. [%]	Param #	FLOPS
CNN	96.19	81.43	17,321	17,315
CNN + LSTM	98.49	80.81	51,932	86,621
CNN (Q)	96.13	86.42	17,321	17,315
CNN + LSTM (Q)	97.96	83.16	51,888	69,299
Random Forest	85.8	85.8	-	-

Note: Q is a quantised model

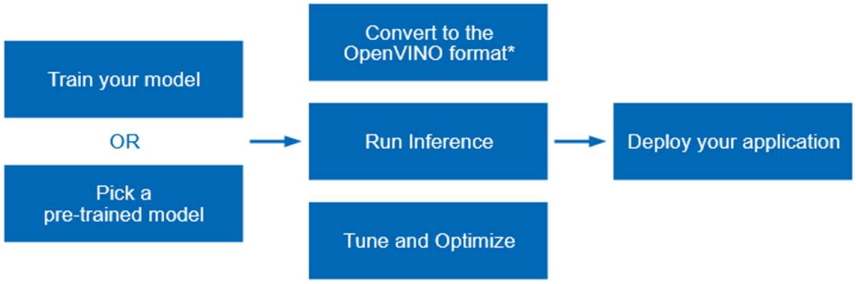
Hardware Estimation:

Before we move forward to the implementation of these models on an embedded device, we used hardware estimation and checked how our models worked on various CPUs and GPUs.

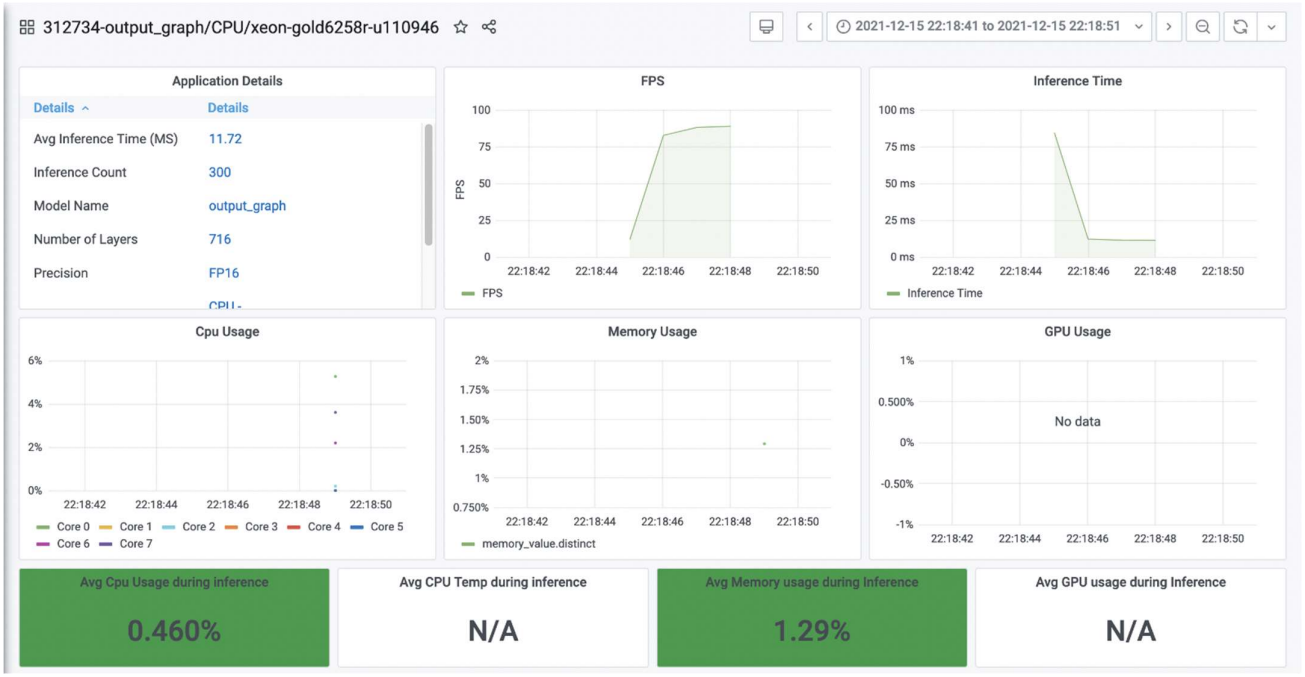
We used Intel's **OpenVino Toolkit** to do the same. There are various boards available on their platform but we used the ones given in the list below.

- Intel Core i5-6500TE CPU
- Intel Core i7-8865UE CPU
- Intel Xeon Gold 6258R CPU
- Intel Xeon E3-1268L v5 CPU
- Intel Core i5-6500TE GPU
- Intel Atom x7-E3950 UP2 GPU

We used our own model which we converted into an OpenVino model for hardware estimation. The python scripts available by OpenVino helped us convert our models.

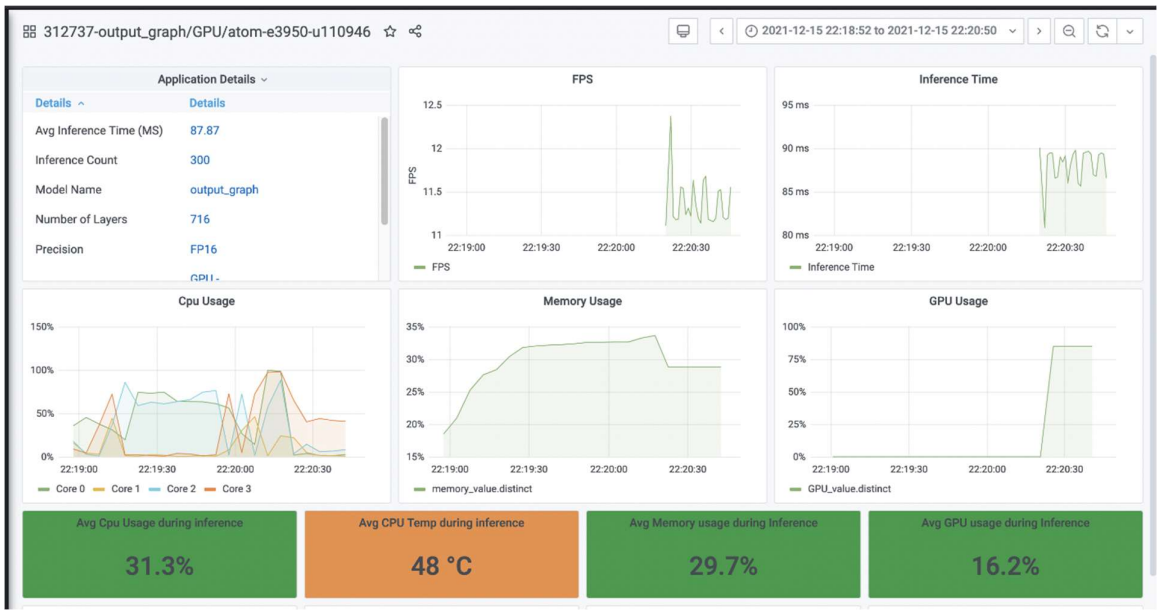


The hardware estimation results are given in the pictures below.




312735-output_graph/CPU/xeon-e3-u110946
2021-12-15 22:18:43 to 2021-12-15 22:19:14
Application Details
Details
Avg Inference Time (MS) 32.31
Inference Count 300
Model Name output_graph
Number of Layers 716
Precision FP16
FPS
2021-12-15 22:19:02
FPS: 9.08
Inference Time
Cpu Usage
Core 0 Core 1 Core 2 Core 3 Core 4 Core 5 Core 6 Core 7
Memory Usage
memory_value.distinct
GPU Usage
GPU_value.distinct
Avg Cpu Usage during inference
18.2%
Avg CPU Temp during inference
35.0 °C
Avg Memory usage during Inference
1.88%
Avg GPU usage during Inference
0%


312736-output_graph/GPU/core-i5-u110946
2021-12-15 22:18:48 to 2021-12-15 22:19:44
Application Details
Details
Avg Inference Time (MS) 34.93
Inference Count 300
Model Name output_graph
Number of Layers 716
Precision FP16
FPS
Inference Time
Cpu Usage
Core 0 Core 1 Core 2 Core 3
Memory Usage
memory_value.distinct
GPU Usage
GPU_value.distinct
Avg Cpu Usage during inference
24.6%
Avg CPU Temp during inference
32.8 °C
Avg Memory usage during Inference
12.0%
Avg GPU usage during Inference
17.2%


312737-output_graph/GPU/atom-e3950-u110946
2021-12-15 22:18:52 to 2021-12-15 22:20:50
Application Details
Details
Avg Inference Time (MS) 87.87
Inference Count 300
Model Name output_graph
Number of Layers 716
Precision FP16
FPS
Inference Time
Cpu Usage
Core 0 Core 1 Core 2 Core 3
Memory Usage
memory_value.distinct
GPU Usage
GPU_value.distinct
Avg Cpu Usage during inference
31.3%
Avg CPU Temp during inference
48 °C
Avg Memory usage during Inference
29.7%
Avg GPU usage during Inference
16.2%