



Edge AI for ECG Monitoring

EECS 199 – Individual Study

Professor Mohammad A Al Faruque

Vishwanath Singh

Spring 2023

Table of Contents

<u>Introduction</u>	3
<u>Edge-AI</u>	3
<u>Esp32 and AD8232</u>	4
<u>Electrocardiogram</u>	5
<u>ECG Data Acquisition and Recording</u>	7
<u>Dataset for the Model</u>	7
<u>Preprocessing</u>	8
<u>1D Convolutional Neural Network</u>	11
<u>Training and Testing</u>	14
<u>Converting Model Into Header File</u>	16
<u>Improvements</u>	18
<u>Summary and Result</u>	18
<u>References</u>	20

I. Introduction

Heart disease continues to be the leading cause of death in the United States and constitutes an ongoing struggle for public health professionals nationwide. Although multiple factors contribute to this dire reality. Expensive healthcare represents one major hindrance for widespread prevention and early detection efforts concerning cardiac conditions. In light of this issue. Our project seeks to design an Internet of Things (IoT) device empowering individuals with accessible yet efficient monitoring capabilities over their heart health status. At its core. This novel initiative relies upon two pivotal components: IoT technology strategically coupled with cutting edge artificial intelligence framework aimed at addressing issues specific to edge environments. To delve into specifics. Edge AI deals with the deployment of powerful machine learning models onto microcontrollers located at the edge of IoT networks. These devices possess limited functionalities but remain capable of performing crucial classification and predictive tasks based on input data.

Leveraging edge AI allows us to unlock the massive potential offered by IoT devices for pervasive analysis and inference without relying excessively on resource hungry cloud-based systems.

To collect relevant physiological data we employ a three lead sensor that can effectively capture a persons electrocardiogram signals for heart function measurements. Those signals feed into our specialized one-dimensional convolutional neural network (1 D CNN) model developed to assess heart health.

Once this model is trained and fine tuned. We streamline its transferability via creating a header file. This file enables simple deployment on minimal resource processor units such as ESP32 while maintaining performance efficiency and accuracy.

By integrating IoT, edge AI, and ECG analysis, this project aims to democratize access to heart health monitoring. Enabling individuals to gain insights into their cardiac well-being through a portable and cost-effective device has the potential to revolutionize healthcare. This report delves into the implementation details, challenges encountered, and the significance of deploying machine learning models at the edge for personalized health monitoring.

II. EDGE-AI

A. What is edge AI?

Edge AI is the integration of artificial intelligence in the real world and as close as possible to the user. The closest IoT layer to the user is the edge layer. Since AI computing is done at the edge of the network and close to where the data is located it is called the edge artificial intelligence. Performing computation at the edge is heavy, but it saves time and energy in sending data to a suitable location for computation to take place. [2]

The benefits of edge AI are intelligence. AI applications are more powerful than conventional applications that need user input. It can be trained how to deal with a particular set of problems rather than learning how to deal with one specific problem. The second is reducing cost. The computation taking place close to data reduces the need for higher bandwidth which reduces

network cost massively. Edge AI is capable increases privacy and availability by handling the data locally and without any human intervention.

B. challenges

The machine learning models are written in high level languages and are compute heavy, they cannot easily be deployed on to a microcontroller since edge devices support languages like C. Edge devices are limited in computational resources, storage and may be in dynamic or unreliable network conditions. But these factors can be targeted in multiple and innovative ways. [3]

C. How are we deploying the model

We are going to train the model in python on a computer which then with the help of EloquentTinyML library which runs tensor flow lite models can be deployed into a microcontroller such as an ESP-32. “TinyML is the overlap between Machine Learning and embedded (IoT) devices”. [4]

TinyML is a new concept, first mentions are dating back to 2018. For complex use cases where you cannot use only logic, ML algorithms can run on low powered devices on the edge.

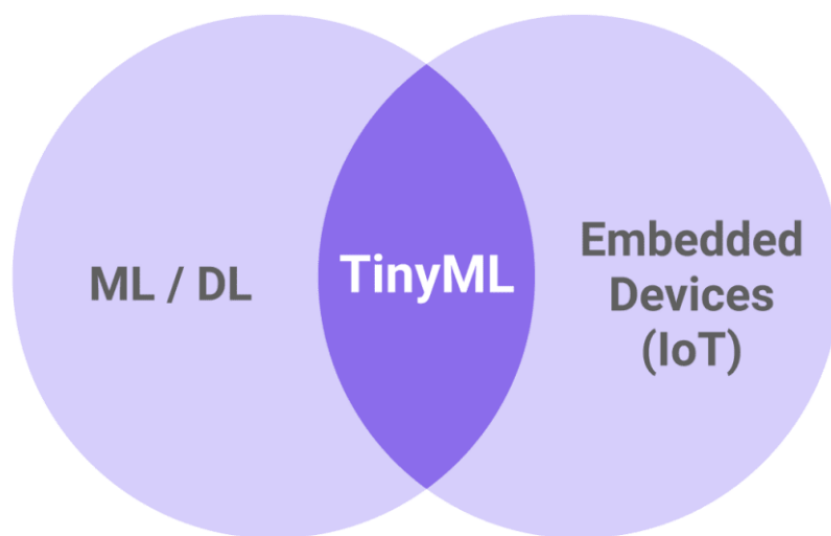


Figure 1: TinyML

Tiny ML uses less bandwidth, decreases the latency, reduces the cost because the data is not sent to the cloud. It also increases the reliability, since data will not be lost in transit to a compute station.

III. ESP-32 AND AD8232

The ESP32 microcontroller serves as an ideal choice for this project due to its affordability and low power consumption. To visualize the electrocardiogram (ECG) signal, we utilized the Arduino IDE, which provides a user-friendly interface. Additionally, the ESP32 is compatible with Micro Python, which allowed us to experiment with implementing small-scale machine learning models. As we encountered challenges when running TensorFlow models on the microcontroller, we explored an alternative approach using TinyML, as mentioned earlier. This decision enabled us to overcome the limitations and successfully deploy a 1-D convolutional neural network (CNN) model for ECG analysis on the ESP32 microcontroller.

AD8232 sensor is used for ECG and biopotential measurements. It detects the ECG signal of the heart using 3-led electrodes connected to your right and left arm and your right leg. It had capabilities to extract and filter noise if it picks up any.

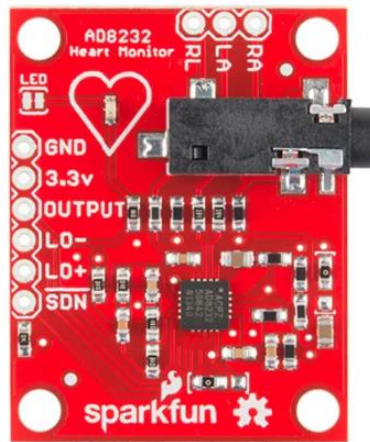
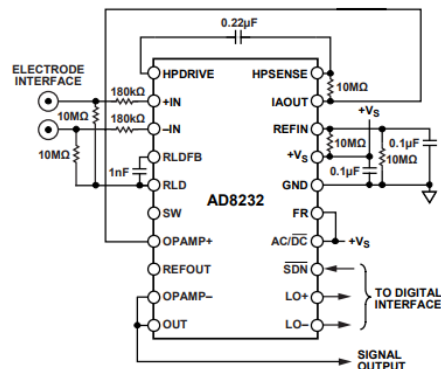


Figure2: AD8232 sensor

The electrode half-cell potential and motion distortions can be filtered out using the AD8232's two-pole high-pass filter. This filter allows for high-pass and massive gain filtering in a single stage, saving both money and space. It is tightly linked with the amplifier's instrumentation architecture.

To remove further noise, the AD8232 may create a three-pole low-pass filter using an uncommitted operational amplifier. Every filter has a frequency cutoff that can be altered by the user to suit different needs. [5]



[5]

IV. ELECTROCARDIOGRAM

In an electrocardiogram (ECG), the P wave signifies the contraction of the atria. The PR interval measures the time from the beginning of atrial contraction to the start of ventricular contraction. The QRS complex represents the complete ventricle depolarization, indicating their contraction.

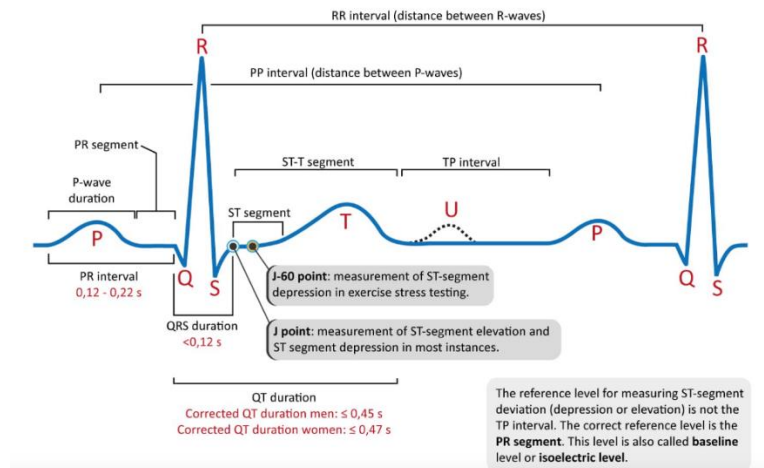


Figure 4: Normal ECG [6]

The ST segment in an electrocardiogram (ECG) represents ventricular depolarization. Changes in the elevation or depression of this segment can potentially indicate ischemia in the heart muscle.

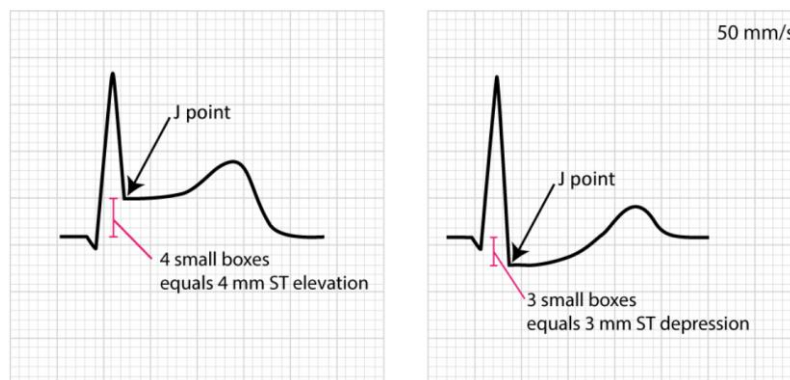


Figure 5: ST elevation and depression [6]

The QT interval in an electrocardiogram (ECG) signifies the entirety of ventricular depolarization and repolarization. When the QT interval is longer than usual, it becomes a risk factor for ventricular arrhythmias and sudden death.

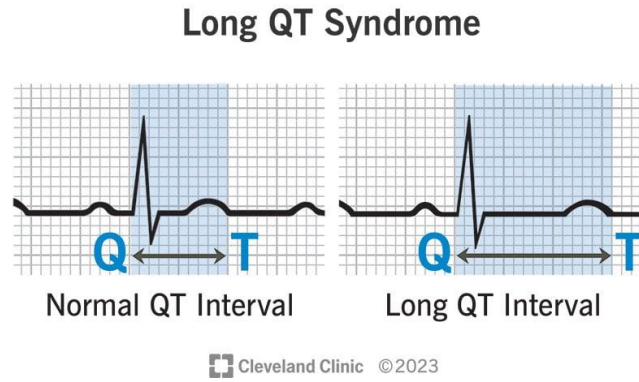


Figure 6: Prolonged QT interval [7]

Having understood the crucial underlying principles and of an electrocardiogram (ECG) signal will help in the successful implementation of our project. While it is important to note that the AD8232 sensor used in our project may not provide clinical-grade accuracy in capturing ECG signals, it serves as a valuable tool for gaining insights and improving our understanding of how future products can be developed. By working with the AD8232 sensor, we can refine our techniques and explore potential enhancements to create more advanced and precise ECG monitoring solutions in the future.

V. ECG DATA ACQUISITION AND RECORDING

In order to get accurate and reliable ECG signal for the project so as to extract features to be able to input into the model, we need to make sure that the 3 Lead placement of the electrodes is correct and does not have any noise and other interference during reading. While the sensor is not clinical grade, it is the most affordable for this project. This sensor would be the starting point in all the improvements that can be made on this project.

When we follow the guidelines for lead placement, we see that 3 three electrodes need to place on the left hand, right hand and the left leg. This lead placement helps read the heart at an angle of 120 degrees. Once the lead is attached, the sensor starts detecting and measuring the electrical activity of the heart. The acquired ECG signal was then transmitted to our data acquisition system.

To make sure that the data is reliable, we use a raspberry pi 3B+ to be interfaced with the AD8232 sensor. The raspberry pi was programmed to sample the ECG signal at a predefined frequency and store it as a csv file.

Due to hardware noise and sensor limitations the data recorded by the AD8232 sensor was not accurate enough to be used as a test case. Therefore, the alternative is to use samples from the dataset used to create the model.

VI. DATASET FOR THE MODEL

The dataset utilized in this project is the ECG Arrhythmia Classification Dataset, which combines four major datasets to extract ECG features for arrhythmia detection. The dataset incorporates ECG features from the 2-Lead signal obtained from the MIT-BIH Supraventricular Arrhythmia Database, MIT-BIH Arrhythmia Database, St Petersburg

INCART 12-lead Arrhythmia Database, and Sudden Cardiac Death Holter Database. The MIT-BIH and Sudden Cardiac Death Holter databases are sourced from the Physionet databank. Each of these databases contributes 17 features, encompassing both 2-Lead and 5-Lead ECG features. The dataset encompasses five classes: normal, supraventricular ectopic beat, ventricular ectopic beat, fusion beat, and unknown beat. A total of 75 subjects are included, with their ECG signals recorded and features extracted for analysis. [8] [9]

The MIT-BIH supraventricular arrhythmia database is a collection of 78 half-hour ECG recordings specifically selected to enhance the representation of supraventricular arrhythmias. These recordings serve as valuable additions to the examples of supraventricular arrhythmias available for analysis. Supraventricular tachycardia occurs when the sinoatrial node and the atrioventricular node do not shoot synchronously, causing an irregular heart rate which begins above the 2 ventricles. [10]

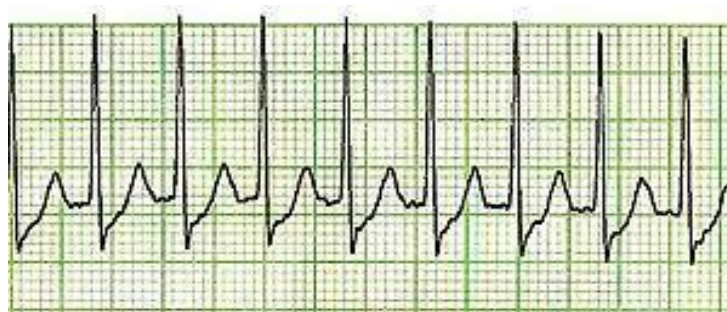


Figure 7: Supraventricular tachycardia

Source: Wikipedia Images

On the other hand, the MIT-BIH Arrhythmia Database comprises 48 half-hour excerpts of two-channel ambulatory ECG recordings. These recordings were obtained from 47 subjects who were studied by the BIH Arrhythmia Laboratory during the period between 1975 and 1979. The dataset provides a historical perspective and a comprehensive set of ECG recordings for research and analysis purposes. [11]

Ventricular tachycardia is an irregular heart rhythm which occurs when the ventricles do not pump blood synchronously, causing the body to not receive enough oxygenated blood.

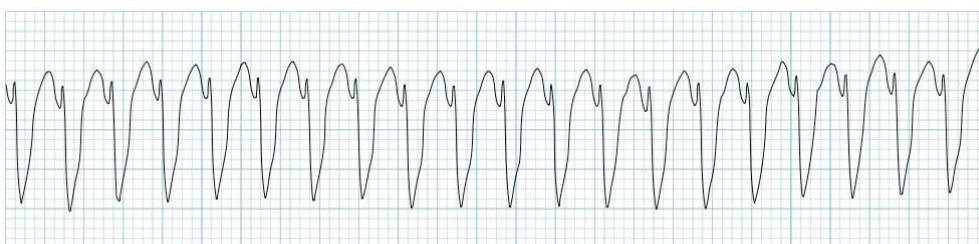


Figure 8: Ventricular Tachycardia

Fusion beats occur when a sinus and ventricular beat coincide to produce a hybrid complex of intermediate morphology.

The Sudden Cardiac Death Holter databases is a collection of long-term ECG signals of patients who had sudden cardiac death during the recordings. [12]

VII. PREPROCESSING

The primary objective of developing this model is to classify input data as either normal or abnormal signal or heart rhythm. This classification process serves as the initial step in evaluating the model's performance on an ESP32 microcontroller, which possesses limited memory capacity. When considering the limitations of computing power, it is beneficial to use categorical classification as it requires less computational resources. By doing so, we keep in mind the restrictions on storage and computing capacity of the Esp32 while still being able to classify data.

To create our model, we rely on multiple Python libraries. Specifically, we use the Keras library for deep learning, which provides a wide range of classes and functions that are essential to our work. We employ Sequential to build our model layer by layer in a sequential flow of data. Additionally, Conv1D allows us to extract meaningful features from input data in a 1D convolutional layer in a neural network. Along with MaxPooling1D class operating 1D max pooling operation reducing dimensionality of the data while retaining relevant information helps us prepare it for analysis. Flatten class is employed as well Transforming Input Data into vector-compatible with rest of layers ensuring reliable efficiency Dense class these libraries offer us all necessary tools and utilities required for building an effective machine learning model.

We extracted 34 features from both 2-Lead and 12-Lead ECG signals used in our dataset. This dataset preparation will enable us to effectively classify all classes as binary classifiers without ambiguity or inconvenience simplifying further analysis. However, since we focus only on 2-Lead signals, we exclude the remaining columns starting from the 18th column. By dropping these unnecessary columns, we streamline the dataset and retain only the relevant features for our classification.

Following this step, we proceed to count the occurrences of different types of classes present in the data, such as normal, ventricular arrhythmia, etc. This helps us gain insights into the distribution of different class types within the dataset. It prepares us to classify all the classes into a binary classification.

In the next step, we consider all data points labelled as supraventricular, ventricular, fusion, and unknown as abnormal data. This decision is based on the goal of performing binary classification on the dataset, where the main focus is to differentiate between normal and abnormal instances. By treating these specific label types as abnormal, we establish a binary classification framework that allows us to effectively categorize the data into normal and abnormal classes.

The values for all the classes are as follows:

Normal	20000
--------	-------

Ventricular arrhythmia	153546
Supraventricular arrhythmia	1958
Fusion	219
Unknow beats	6

Table 1: Multi Class Values

After calculating the different number of values in all the 5 classes, we transform the multiclass label into binary class label as discussed before.

Normal	153546
Arrhythmia	22183

Table 2: Binary Class Values

It is clear from the values in the table that there is a class imbalance in the dataset. When the dataset has a notable differential in the quantity of data points between several classes, it is said to be overfitting or class imbalance. Analysing this context shows us that there's an underrepresentation of instances within the minority group concerning the majority class present in our dataset. Due to this imbalance, any classification procedure based on solely one dimension will tend toward favouring one category over another as more population identifies with it. It's necessary that any efforts done regarding model creation include taking into consideration these imbalances as without them, skewed results can be perceived — resulting in biased decisions.

Handling such cases, strategies like oversampling through Synthetic Minority Over-Sampling Technique(SMOTE), performing undersampling or using pre-made algorithms based on balancing methods are popular gained traction counting numerous success stories behind their back-logics.

A pre-eminent solution among those would be handling oversample within sections underrepresented compared against minorities within a dataset by creating synthetic cases mimicking these categories' characteristics through neighbouring closeness sampling belonging inside each associated feature space while adding this generated data in corresponding volume.

To apply this approach in our project, we can import SMOTE classes and implement oversampling techniques until obtaining an exact balance between minority and majority groups within the dataset. This should lead to better-suited training models that provide accurate predictions.

The values of the classes are displayed in table 3 below.

Before (175729, 1)	
Normal	153546

Arrythmia	22183
After (307092, 1)	
Normal	153546
Arrythmia	153546

Table 3: Binary Class Values after resampling using SMOTE.

As we use machine learning algorithms, proper preprocessing techniques are invaluable in preparing datasets for optimal model performance and reliability. Using scikit-learn library functions, such as `train_test_split()`, we divided our dataset into separate training and testing subsets before applying normalization using `MinMaxScaler()` via `fit_transform()`. The benefits of normalizing each feature value assists in maintaining directly comparable scales regardless of their initial ranges.

We then employed `transform()` when applying normalization parameters to testing data which were derived beforehand from our fitting routine on training data while producing insight into scaled feature range variations among features leveraging `min_max_scaler.scale_` print statements.

Consequently, essential among these steps was modifying categorical labels such that they could be translated into binary labels for better model compatibility using `OneHotEncoder` accurately provided by scikit-learn respectively. An essential assessment of the usefulness of `OneHotEncoder` in this scenario was crucial. `OneHotEncoder` is specifically employed to address this issue where ML models require numeric input instead of categorical variables. The numeric labels are more effective for training and prediction, if we directly convert the labels it may cause ordinal relationships to form that actually don't exist in the main data. Therefore, it transforms categorical labels into a binary vector representation, where each unique category is represented by a binary feature column. In other words, it creates a "one-hot" encoding for each category, indicating the presence or absence of a specific category using binary values (1 or 0).

By using `OneHotEncoder`, we ensure that the categorical labels are encoded in a way that preserves their distinctiveness without introducing any ordinal relationships. This encoding is particularly useful when the labels do not have a natural ordering or when they are not inherently numerical.

VIII. 1D CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Network is an area of deep learning that specializes in pattern recognition. We use 1D CNN since time-series data requires kernel sliding in only one dimension and have spatial properties and ECG signal is a non-stationary time-series data with irregularities in waveform. [15]

Let's understand what convolution is. Convolution is a mathematical operation that takes 2 functions f and g and produces a third function $(f * g)$, and it shows how one function affects the shape of the other function. [16]

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau.$$

Convolutional layers combine the input and send the resulting information to the following layer. This is comparable to how a neuron in the visual brain might react to a particular stimulus. The ability of a CNN to automatically learn and extract pertinent features from unprocessed input data is its primary feature. Convolutional layers, which CNNs incorporate, are specialized layers that apply filters to input data, unlike traditional neural networks. The network can detect patterns as well as characteristics at various spatial hierarchies thanks to these filters. [16]

The fundamental components of a CNN are convolutional layers, pooling layers, and fully connected layers. Filters in the convolutional layers convolve over the input data through a technique known as convolution to capture important spatial information. By reducing sampling of the spatial dimensions of the feature maps and reducing computational complexity, the pooling layers retrieve the most crucial data. Fully connected layers oversee classification or regression tasks by connecting all of the extracted features to the output layer. These layers connect all the recovered characteristics to the output layer. [16]

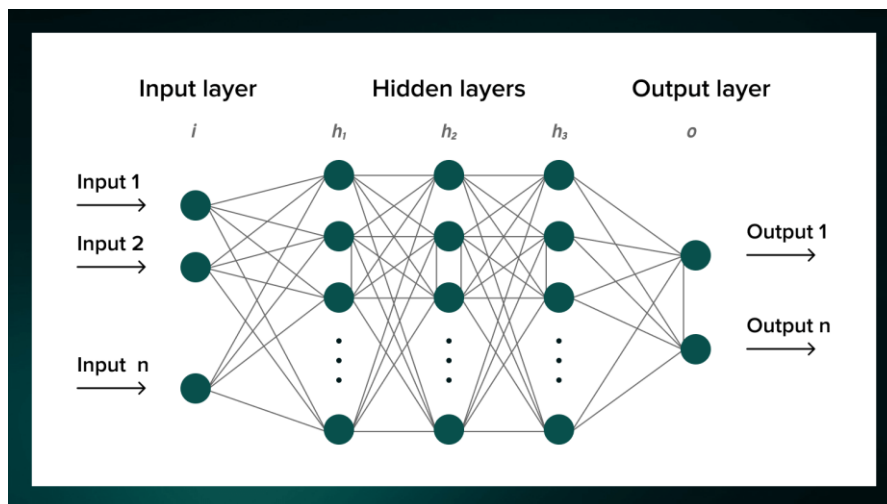


Figure 9: Convolution Neural Network [serokell.io]

In the process of creating our model, we start by initializing a sequential model using the Keras library. The sequential model represents a linear stack of layers where each layer is added one by one.

To move forward we must introduce our first convolutional layer - a critical component for constructing meaningful features from our input data. Through a process called convolution,

learnable filters are applied to our inputs in order to function as "feature detectors" or "templates" that can identify patterns. As these filters are passed over our inputs they highlight different areas which allow us to capture local patterns and spatial relationships among other things -- all relating back to identifying key features like corners, edges and textures within the dataset . Additionally by comparing learned filters across many different regions of input-data , it's likely that local features get extracted which translate into more overall meaningful representations for not only subsequent layers involved but also possibly whatever ambitious real-world task you may have ahead.

```
#1D CNN on each col
model = Sequential()
model.add(Conv1D(64, kernel_size=3, activation='relu', input_shape=(16, 1)))
model.add(Conv1D(64, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(128, kernel_size=3, activation='relu'))
model.add(Conv1D(128, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(2, activation='softmax'))

# Step 5: Model Training
# Compile and train the VGG model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])#
```

Figure 10: CNN model for the dataset

The model we are examining contains several layers that perform different tasks in order to analyse the data fed into the system. The first layer is a convolutional layer with 64 filters and a kernel size of 3 that uses Rectified Linear Unit (ReLU) as an activation function. ReLU sets negative values to zero while preserving positive values allowing the network to learn non linear relationships in the data. The second layer is similar to the first convolutional layer. But without an input size parameter. It further extracts features from the input data using the same configuration as its predecessor.

After these convolutional layers. We introduce a pooling layer that combines outputs from multiple neurons into a single neuron in the subsequent layer. This reduces the dimensionality of the data and helps achieve better computations with faster processing times. Here our pool size is set at 2, where only maximum values in each 2x2 region are considered while spatial dimensions are halved. We then have two additional convolutional layers added with 128 filters each that help identify more complex patterns and improve subsequent mapping layers' output connections.

Finally, we have included a flattening layer that transforms previous layer outputs into a one dimensional vector before connecting it with fully connected layers for an efficient learning process. The final layer is a dense layer, which consists of two dense layers. The first dense layer has 256 neurons, followed by the second dense layer with 2 neurons. The activation

function used in these dense layers is SoftMax, which provides probabilities for each class in the classification task. SoftMax ensures that the output values represent the likelihood of the input belonging to each class, enabling the model to make class predictions.

Overall, this model architecture combines convolutional layers for feature extraction, pooling layers for dimensionality reduction, and dense layers for classification. The use of ReLU activation, increasing the number of filters, and employing SoftMax activation in the final layer are key design choices aimed at improve the model's ability to learn and classify data accurately.

IX. TRAINING AND TESTING OF THE 1D CONVOLUTIONAL NEURAL NETWORK MODEL

We noted earlier that splitting data into separate groups is essential for effective model tuning and evaluation. In line with this principle, we utilize scikit-learn's `train_test_split` function to create two distinct datasets: one for learning (training) algorithms, whiles running predictive analysis on another for generalization performance evaluation (testing). As is often done in most machine learning applications, a partition ratio with 75%/25% allocation respectively is used by default where test size accounts for a quarter of all samples.

For optimized AI-models improving their generalization capabilities remains vital - splitting a comprehensive training dataset into two subsets creates an excellent option for scaling up. The two sets consist of custom-built training-bound options that allow far-reaching changes exploration based on large input data volumes and decoding layers. The line: `"model.fit(X_train_scaled[0:n, :], encoded_labels[0:n], batch_size=32, epochs=10, validation_data=(X_train_scaled[n:, :], encoded_labels[n:]))"` uses this particular split-subset innovation to enhance optimum effectiveness in driving more in-depth analysis for producing highly accurate predictions. To refine accuracy under this approach there is also an iteration process – after each epoch of the subset-based run, evaluations are done using a validation-set from a subset of the same training set used earlier. After iteration completion on said subsets to optimize accuracy based on traditional loss functions and prediction accuracy checks., we leverage testdata sets inputs represented by `X_test` into our pre-tuned AI-system applying the encoding-label's algorithm - resulting predictively outputs matched against desired results consigned to encoded labels.

Following an assessment, our obtained results show that we received a test loss total of 540.024 combined with an accompanying precision reading of 0.7153.

Test losses are recorded as numerical values that represent potential deviations between predicted versus actual labeling within any given assessed dataset - increased totals suggest wider disparities within these results.

```

(207287, 16)
Epoch 1/10
6478/6478 [=====] - 36s 5ms/step - loss: 0.0768 - accuracy: 0.9744 - val_loss: 0.0422 - val_accuracy: 0.9877
Epoch 2/10
6478/6478 [=====] - 35s 5ms/step - loss: 0.0434 - accuracy: 0.9872 - val_loss: 0.0406 - val_accuracy: 0.9875
Epoch 3/10
6478/6478 [=====] - 33s 5ms/step - loss: 0.0367 - accuracy: 0.9894 - val_loss: 0.0344 - val_accuracy: 0.9903
Epoch 4/10
6478/6478 [=====] - 34s 5ms/step - loss: 0.0330 - accuracy: 0.9904 - val_loss: 0.0330 - val_accuracy: 0.9904
Epoch 5/10
6478/6478 [=====] - 36s 5ms/step - loss: 0.0304 - accuracy: 0.9911 - val_loss: 0.0263 - val_accuracy: 0.9924
Epoch 6/10
6478/6478 [=====] - 34s 5ms/step - loss: 0.0284 - accuracy: 0.9917 - val_loss: 0.0299 - val_accuracy: 0.9913
Epoch 7/10
6478/6478 [=====] - 34s 5ms/step - loss: 0.0266 - accuracy: 0.9923 - val_loss: 0.0269 - val_accuracy: 0.9914
Epoch 8/10
6478/6478 [=====] - 33s 5ms/step - loss: 0.0253 - accuracy: 0.9926 - val_loss: 0.0238 - val_accuracy: 0.9929
Epoch 9/10
6478/6478 [=====] - 33s 5ms/step - loss: 0.0240 - accuracy: 0.9929 - val_loss: 0.0250 - val_accuracy: 0.9924
Epoch 10/10
6478/6478 [=====] - 35s 5ms/step - loss: 0.0228 - accuracy: 0.9933 - val_loss: 0.0194 - val_accuracy: 0.9937
<keras.callbacks.History at 0x7f33fd1edbd0>

```

Figure 11: 10 Epoch of 6478 samples

When evaluating trained models we use methods such as "evaluate()" which computes both losses and accuracies in relation to any given dataset; specifically here X_test serves as our input feature combination whereas appropriately encoded-labels represent their corresponding values.

After the evaluation is performed, the obtained test loss is 540.024 and the test accuracy is 0.7153. The test loss is a numerical value that represents the discrepancy between the predicted labels and the true labels in the testing dataset. A higher test loss indicates a larger deviation between the predicted and true labels.

The test accuracy, on the other hand, measures the percentage of correctly predicted labels out of all the samples in the testing dataset. In this case, the model achieved a test accuracy of 0.7153, which means that approximately 71.53% of the labels in the testing dataset were correctly predicted by the model.

The obtained test loss and accuracy values can be used to assess the effectiveness of the trained model and compare it with other models or baselines.

```

loss, accuracy = model.evaluate(X_test, encoded_labels)
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)

2400/2400 [=====] - 4s 2ms/step - loss: 540.0249 - accuracy: 0.7153
Test Loss: 540.02490234375
Test Accuracy: 0.7153295874595642

```

Figure 12: Test accuracy and loss after validation

The confusion matrix is a crucial tool for evaluating a classification model's performance and accuracy. The model's predictions are broken out in great depth, with the counts of true positives, true negatives, false positives, and false negatives displayed. We can learn more about the model's precision in classifying objects by looking at these values.

True positives and true negatives are situations in which the model accurately predicted the positive class and the negative class, respectively. False negatives happen when the model

The confusion matrix gives us a complete picture of the model's accuracy and lets us assess how well it performs across all classes. It enables us to comprehend the model's advantages and disadvantages, spot areas that could use improvement, and assess the relative effectiveness of other models and algorithms.

X. CONVERTING OUR MODEL INTO A HEADER FILE

```
from tinymlgen import port
# Convertir modelo en un array C
c_code = port(model, pretty_print=True)
print(c_code)
```

The generated code is output and can be seen in figure 13.

16

Figure 13: Generated C code saved as sensor model.h

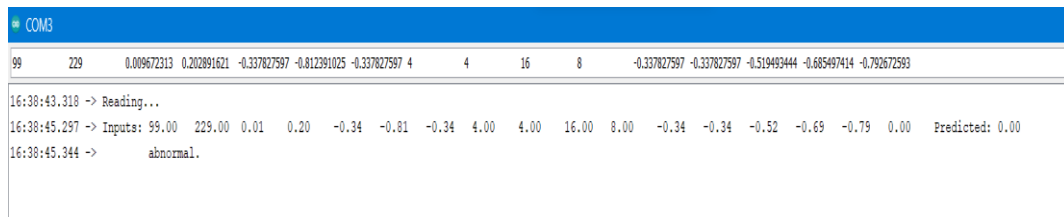
Next, we call the sensor model.h into a C code named sensor prediction. The code starts by including the necessary libraries for running TensorFlow models on microcontrollers. It then defines some constants that specify the number of input features, the number of output values, and the size of the memory allocated for TensorFlow operations.

The input values are given into the serial monitor and the predicted output are then printed for debugging purposes. Based on the value of predicted, a message indicating whether the prediction is "normal" or "abnormal" is printed.

In summary, the code sets up the microcontroller to receive input data, passes the data through a TensorFlow model, and provides the predicted output along with a corresponding message indicating its interpretation.

We take 1 data reading from the dataset and input it to the serial monitor of the Arduino IDE. Once the values are given, it is sent to the model and the output is displayed indicating if the data entered shows any signs of arrhythmia or not.

The code used in this case is from the EloquentTinyML repository which was then modifies for my model and the input values that were given to the model.



```
* COM3
99      229      0.009672313  0.202891621  -0.337827597  -0.812391025  -0.337827597  4      4      16      8      -0.337827597  -0.337827597  -0.519493444  -0.685497414  -0.792672593

16:38:43.318 -> Reading...
16:38:45.297 -> Inputs: 99.00  229.00  0.01  0.20  -0.34  -0.81  -0.34  4.00  4.00  16.00  8.00  -0.34  -0.34  -0.52  -0.69  -0.79  0.00  Predicted: 0.00
16:38:45.344 ->      abnormal.
```

Figure 14: Output tested on the ESP32 microcontroller.

Unfortunately, the code sensor prediction code does not give the right output every single instance. This could be due to a threshold value not being accurate. Another reason could be there are not enough datapoints to classify the normal ECG signal.

XI. IMPROVEMENTS

Access to a high-quality ECG sensor is vital for capturing accurate ECG signals in real-time and facilitating the application of advanced signal processing techniques. One such technique, Waveform Segmentation Using Deep Learning, can be employed for signal processing and time-frequency analysis using LSTM (Long Short-Term Memory). In a study, by the implementation of bandpass filtering and Fourier-based synchro squeezing resulted in an average improvement in classification accuracy across all output classes from 55% to approximately 85%.[16]

However, during the project, the irregular nature of the ECG signals recorded using the AD8232 sensor posed challenges in implementing the desired signal processing techniques. Despite the limitations encountered, valuable insights were gained into the complexities involved in processing and analysing ECG signals.

Moving forward, an avenue for further improvement lies in developing a multi-class classification model. Such a model would enhance the understanding of various types of arrhythmias a subject may be experiencing, providing a more comprehensive approach to monitoring heart health. This advancement in classification capabilities has the potential to revolutionize heart health monitoring practices worldwide, enabling more accurate diagnoses and personalized interventions.

XII. SUMMARY

We embark on this project with the objective of creating an accurate model for classifying electrocardiogram (ECG) signal into either normal or abnormal categories using Convolutional Neural Networks (CNN). The dataset consists of 34 unique features extracted from multiple ECG databases like MIT-BIH Arrhythmia Dataset, MIT-BIH Supraventricular Arrhythmia Dataset and Sudden Cardiac Death Holter Database while considering two-lead and five-lead ECG signals.

For binary classification purposes, data 'abnormality' is determined based on attributes such as supraventricular, ventricular, fusion or unknown while normal status remains unchanged. However, the lack of balance in class distribution prompted us to employ Synthetic Minority Over-Sampling Technique (SMOTE), which helped us create artificial samples to compensate for unsatisfactory minority records.

Subsequently, the pre-processed datasets were sorted into training and testing sets with a division of the training set into further partitions, the latter recognized for validation. Our model features several layers; convolutional layers activated by Rectified Linear Units (ReLU) with different filter sizes, pooling layers to reduce dimensionality, and dense layers to finalize classification.

Our model after validation on the training set archives an accuracy of 0.7153. Subsequent to which we assess its performance via generation of a confusion matrix. This provides data on true positives, true negatives, false-positives and false-negatives to obtain insights into evaluating accuracy and effectiveness in predicting abnormal ECG.

Additionally, the trained model is converted into C code using the tinymlgcn library, allowing for deployment on low-power microcontrollers.

XIII. RESULTS

The project showed several important results in the classification of ECG signals. After training the Convolutional Neural Network (CNN) model on the pre-processed and balanced dataset, it achieved a test accuracy of 0.7153. This indicates that the model can accurately classify ECG signals as normal or abnormal with a reasonable performance. The evaluation of the model's performance using a confusion matrix provided insights into the true positives (23071), true negatives (31847), false positives (15315), and false negatives (6540), further confirming the effectiveness of the model. The successful conversion of the trained model into C code using the tinymlgcn library enables its deployment on low-power microcontrollers, facilitating real-time classification of ECG signals in resource-constrained environments.

However, these results did not meet the end goal of the project since data extracted from AD8232 was not clear and evident enough to be used and the ECG signal was not processed in real time. Yet, the project demonstrates the potential of CNN-based approaches for ECG signal classification and their applicability in practical healthcare scenarios.

XIV. ACKNOWLEDGMENT

I would like to express my sincere gratitude to Dr. Mohammad Al Faruque, my professor for the EECS 199 individual study course for Prof Al Faruque's guidance, expertise, and support throughout the course. I am truly thankful for the knowledge and skills I have gained under his mentorship, which will undoubtedly shape my future endeavours.

XV. REFERENCES

- [1] G. B. M. a. R. G. Mark, "The Impact of the MIT-BIH," IEEE ENGINEERING IN MEDICINE AND BIOLOGY, p. 45, 2001.
- [2] T. YEUNG, "What Is Edge AI and How Does It Work?," Nvidia, 2022.
- [3] M. O. B. U. D. P. M. A. F. Nafiul Rashid, Lecture: Edge AI - Tutorial, Irvine: Al Faruque Lecture, 2023.
- [4] Tomas, "Dev.to," 31 May 2021. [Online]. Available: <https://dev.to/tkeyo/tinyml-machine-learning-on-esp32-with-micropython-38a6>. [Accessed 5 April 2023].
- [5] Anlog.com, "Heart Rate Monitor Front End Data Sheet".
- [6] E. a. E. learning, "Introduction to ECG interpretation," in Clinical ECG Interpretation, ECG and ECHO learning.
- [7] "Long QT Syndrome (LQTS)," Cleveland Clinic, [Online]. Available: <https://my.clevelandclinic.org/health/diseases/17183-long-q-t-syndrome-lqts>. [Accessed 31 May 2023].
- [8] M. M. F. Z. M. F. Sadman Sakib, "Harnessing Artificial Intelligence for Secure ECG Analytics at the Edge for Cardiac Arrhythmia Classification," in Secure Edge Computing, CRC Press, 2021, p. 17.
- [9] G. B. M. a. R. G. Mark, "The Impact of the MIT-BIH," IEEE ENGINEERING IN MEDICINE AND BIOLOGY, p. 6, 2001.
- [10] "MIT-BIH Supraventricular Arrhythmia Database," 3 August 1999. [Online]. Available: <https://physionet.org/content/svdb/1.0.0/>. [Accessed April 2023].
- [11] "MIT-BIH Arrhythmia Database," Physionet, 2005.
- [12] "Sudden Cardiac Death Holter Database," Physionet, July 2004. [Online]. Available: <https://physionet.org/content/sddb/1.0.0/>. [Accessed April 2023].

- [13] J. Brownlee, "SMOTE for Imbalanced Classification with Python," <https://machinelearningmastery.com/>, 2021.
- [14] P. v.-c. likebupt, "SMOTE," Microsoft , 2021.
- [15] J. L. Y.-J. T. C. W. Y. & F.-Y. L. Kuo-Kun Tseng, "Healthcare knowledge of relationship between time series electrocardiogram and cigarette smoking using clinical records," BMC Medical Informatics and Decision Making volume, 2020.
- [16] "Convolution," Wikipedia.
- [17] S. S. Z. M. F. Sadman Sakib, "Harnessing Artificial Intelligence for Secure ECG Analytics at the Edge for Cardiac Arrhythmia Classification," in Secure Edge Computing, Lakehead University, Canada, 2021, p. 137.
- [18] "Waveform Segmentation Using Deep Learning," Matlab.