Anomaly Detection in Heart Activity

Amro Ghoneim
Ismail El Sharkawy
Zeyad Ali

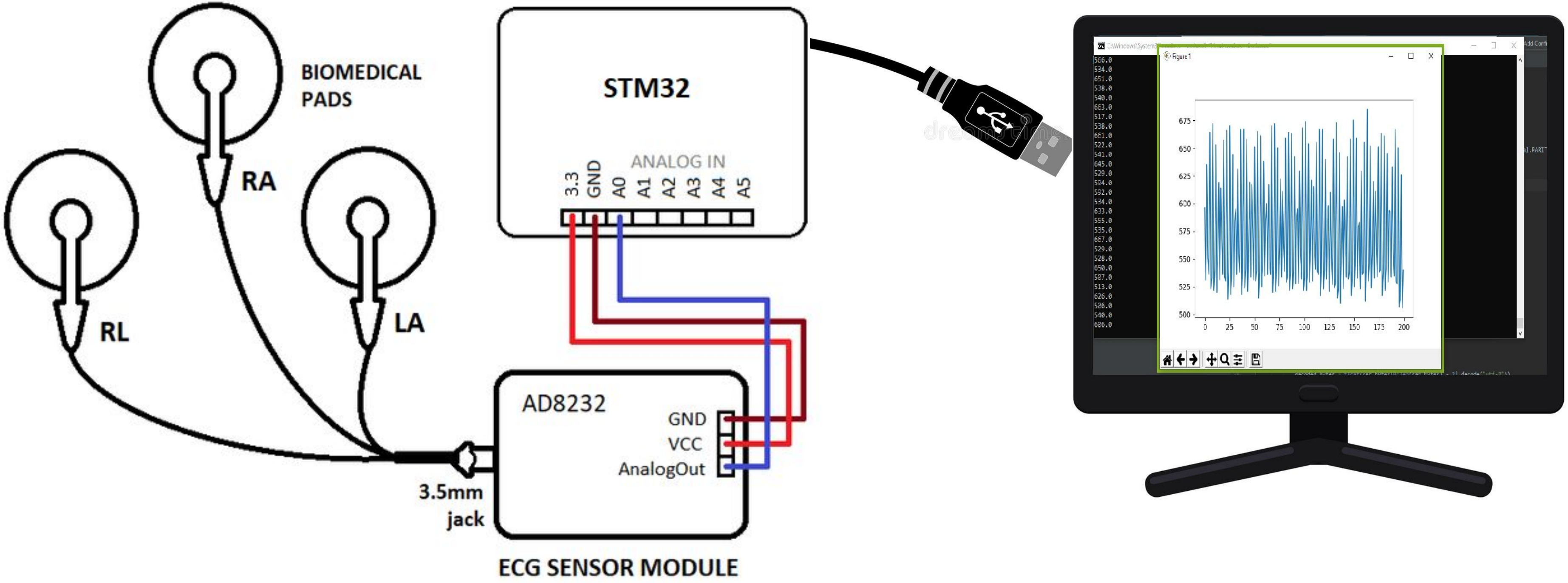
Our Project

The aim of our project is detect anomalies in heartbeats constructed from an electrocardiogram (ECG) monitor, which is the recording of the electrical pulse/activity of one’s heart within the MCU. The recording of the heart beat will be displayed and the system will detect if there is an anomaly in the heart beat.

Design

- The heart pulses will be detected using the AD8232 module.
- The readings detected from the AD8232 will be sent to the STM32 microcontroller on one of its ADC input pins.
- The readings will then be given to the machine learning model deployed on the MCU for real time inference
- The readings/output will then be transmitted via UART and/or displayed on MCU(LEDs) which will be received by a python application.
- The application will will take the readings and display them.

Design


BIOMEDICAL
PADS
RA
RL
LA
STM32
ANALOG IN
3.3
GND
A0
A1
A2
A3
A4
A5
AD8232
GND
VCC
AnalogOut
3.5mm
jack
ECG SENSOR MODULE
Figure 1

Implementation: Keil

The embedded code on the STM32 received the ADC input from the AD8232 and transmitted it through the UART. (Sampling rate unchanged)

```
while (1)
{
    // Test: Set GPIO pin high
  //HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);

  // Get ADC value
  HAL_ADC_Start(&hadc1);
  HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
  raw = HAL_ADC_GetValue(&hadc1);


  // Test: Set GPIO pin low
  //HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);

  // Convert to string and print
  sprintf(reading, "%hu\r\n", raw);
  HAL_UART_Transmit(&huart2, (uint8_t*)reading, sizeof(reading)-5, HAL_MAX_DELAY);

  // Pretend we have to do something else for a while
  HAL_Delay(1);
}
```

Implementation: Python Application

Using python’s library Pyserial, the port of interest “COM3” was specified alongside the corresponding baudrate.

UART output is read line by line and formatted using the decode(”utf-8”) function after being parsed to output desired format.

Data is then gathered and stored in a csv file for later review.

For better visualization of errors and changes, data is then graphed in real time as shown in slide 8.

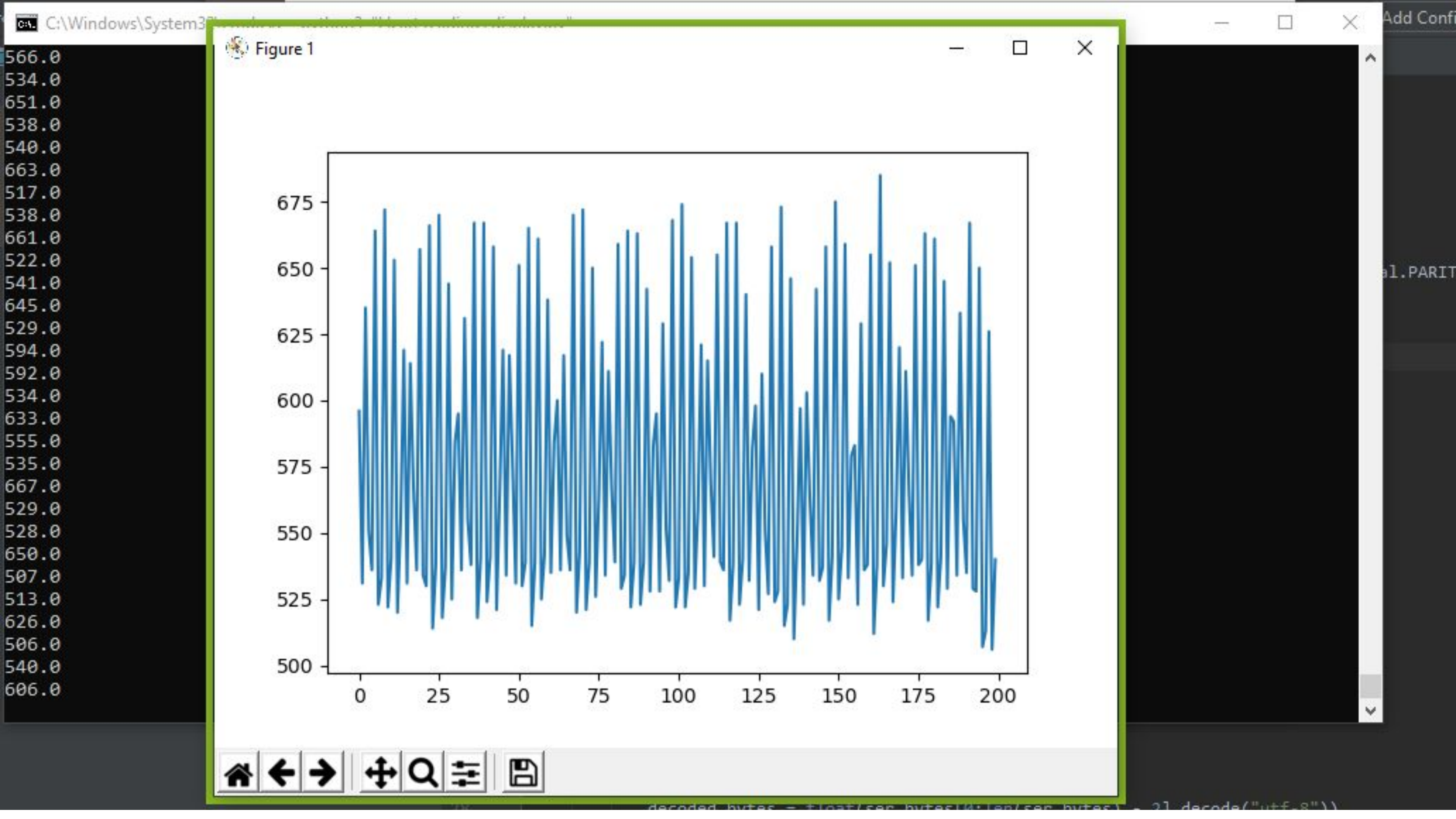
```python
ser = serial.Serial(port='COM3', baudrate=9600, bytesize=serial.EIGHTBITS, parity=serial.PARITY_NONE, timeout=2)
ser.flushInput()
plot_window = 200
y_var = np.array(np.zeros([plot_window]))

plt.ion()
fig, ax = plt.subplots()
line, = ax.plot(y_var)

if ser.isOpen():
    try:
        while 1:
            ser_bytes = ser.readline()
            decoded_bytes = float(ser_bytes[0:len(ser_bytes) - 2].decode("utf-8"))
            print(decoded_bytes)
            with open("test_data.csv", "a") as f:
                writer = csv.writer(f, delimiter=",")
                writer.writerow([time.time(), decoded_bytes])
                y_var = np.append(y_var, decoded_bytes)
                y_var = y_var[1:plot_window + 1]
                line.set_ydata(y_var)
                ax.relim()
                ax.autoscale_view()
                fig.canvas.draw()
                fig.canvas.flush_events()
    except Exception:
        print("error")
```

Looks like you're using NumPy

Results


C:\Windows\System3
Figure 1
566.0
534.0
651.0
538.0
540.0
663.0
517.0
538.0
661.0
522.0
541.0
645.0
529.0
594.0
592.0
534.0
633.0
555.0
535.0
667.0
529.0
529.0
528.0
650.0
507.0
513.0
626.0
506.0
540.0
606.0
675
650
625
600
575
550
525
500
0
25
50
75
100
125
150
175
200
Add Confi
al.PARIT

About the Data set

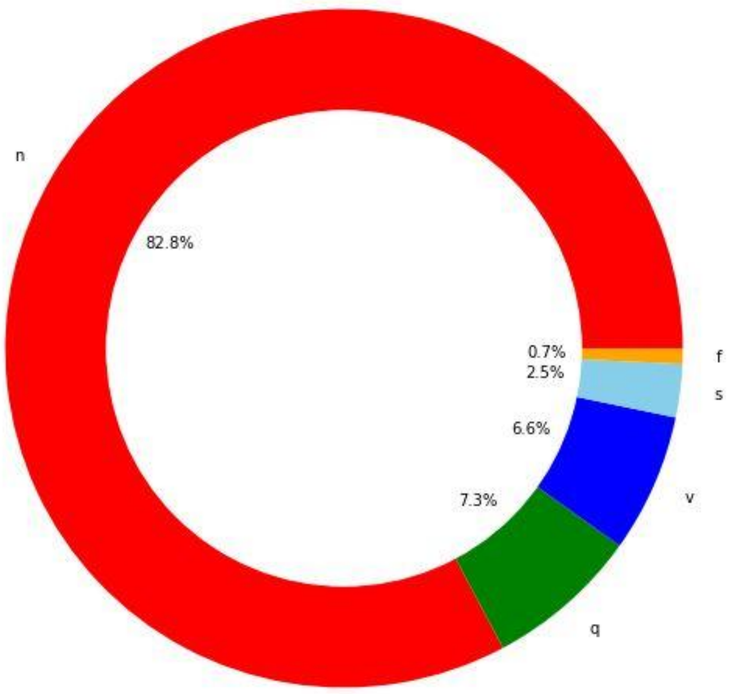
- Number of Samples: 109446
- Number of Categories: 5
- Sampling Frequency: 125Hz
- Data Source: Physionet's MIT-BIH Arrhythmia Dataset
  - Preprocessing steps described in this paper to create the data set
- Classes: ['N': 0, 'S': 1, 'V': 2, 'F': 3, 'Q': 4]

Data set Class Distribution

Initial Class distribution of the data set shows highly imbalanced data !

This will affect model performance for detecting anomalies

```
0    72471
4     6431
2     5788
1     2223
3      641
Name: 187, dtype: int64
```

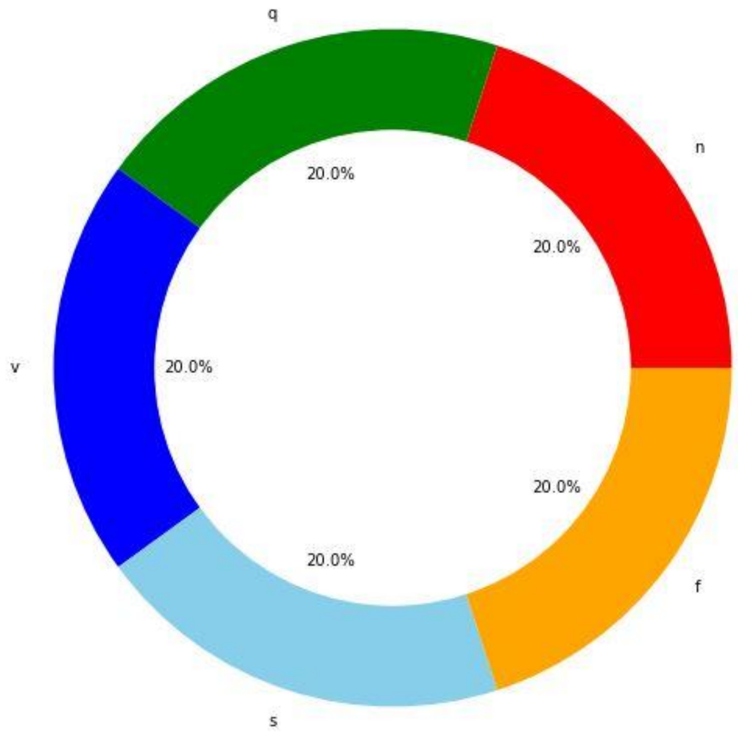


Data set resampling

Resampled the data set to have a balanced class distribution

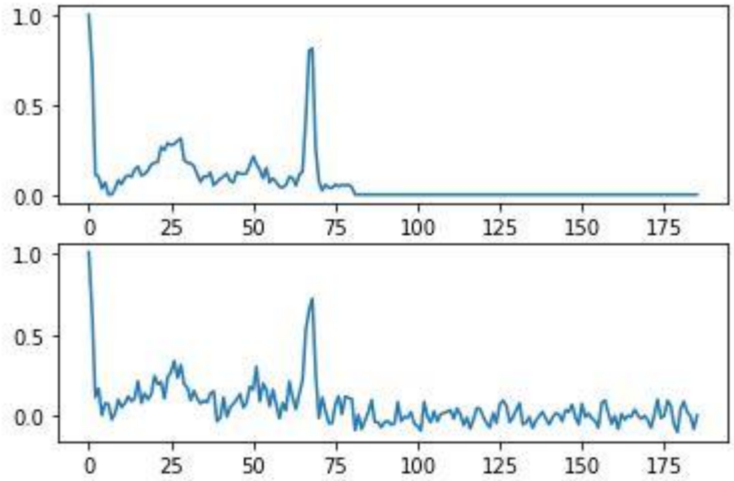
Each class now has 20,000 samples each

```
4    20000
3    20000
2    20000
1    20000
0    20000
Name: 187, dtype: int64
```



Other preprocessing techniques

Added some noise to the data to make it more generalized



Keras

- Open-source neural-network library written in Python

- Can run on top of Tensorflow, Theano and other machine learning frameworks

- User Friendly, Modular and Extensible

Model Architecture

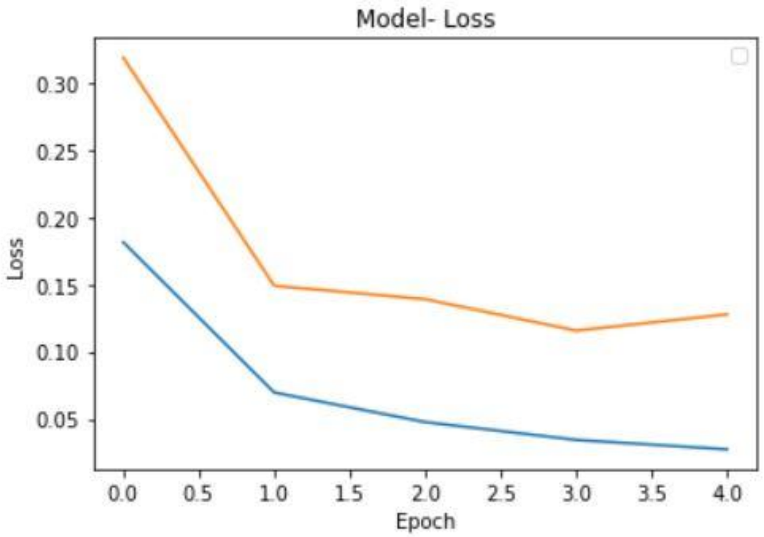
```
data_input=(X_train.shape[1],1)
inputs_cnn=Input(shape=(im_shape), name='data_input')
conv1_1=Convolution1D(64, (6), activation='relu', input_shape=im_shape)(inputs_cnn)
conv1_1=BatchNormalization()(conv1_1)
pool1=MaxPool1D(pool_size=(3), strides=(2), padding="same")(conv1_1)
conv2_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool1)
conv2_1=BatchNormalization()(conv2_1)
pool2=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv2_1)
conv3_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool2)
conv3_1=BatchNormalization()(conv3_1)
pool3=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv3_1)
flatten=Flatten()(pool3)
dense_end1 = Dense(64, activation='relu')(flatten)
dense_end2 = Dense(32, activation='relu')(dense_end1)
main_output = Dense(5, activation='softmax', name='main_output')(dense_end2)
```

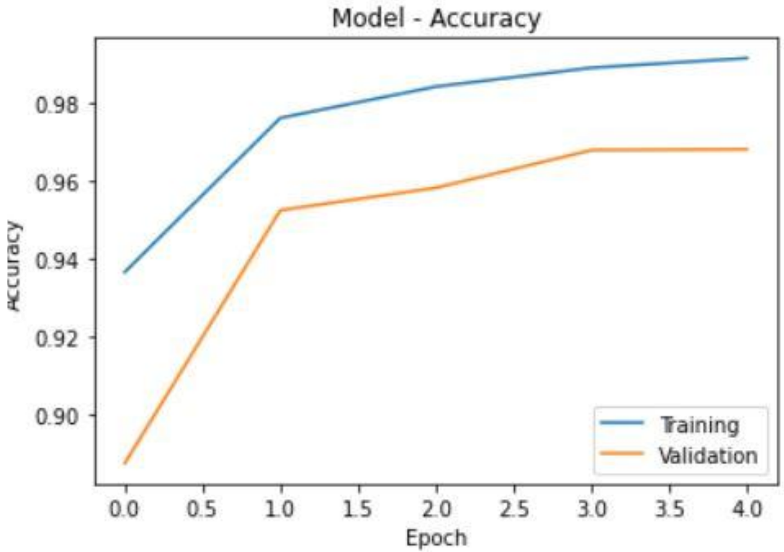
```
model = Model(inputs= inputs_cnn, outputs=main_output)
model.compile(optimizer='adam', loss='categorical_crossentropy',metrics = ['accuracy'])


callbacks = [EarlyStopping(monitor='val_loss', patience=8),
        ModelCheckpoint(filepath='CNN_model.h5', monitor='val_loss', save_best_only=True)]
```

Results

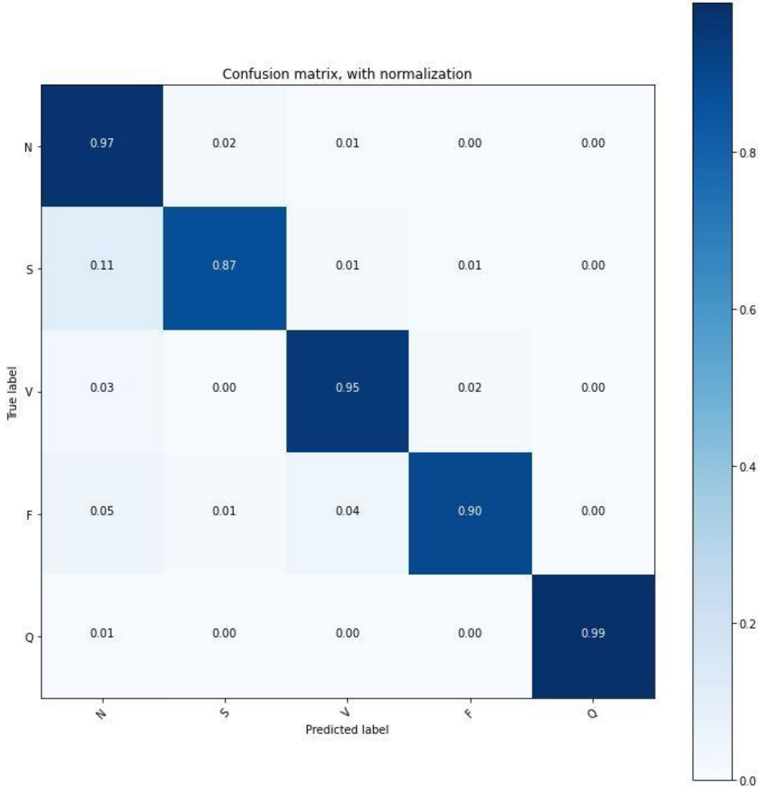
The graphs below show the training **accuracy** and **validation loss** over **5** epochs





Results

- **The confusion matrix shows that the model performs on the test set with slight decrease in results for the two classes S and F**



STM32.AI

- Interoperable with popular deep learning training tools (Keras)
- Compatible with many IDEs and compilers (Keil)
- Sensor and RTOS agnostic
- Allows multiple Artificial Neural Networks to be run on a single STM32 MCU
- Full support for ultra-low-power STM32 MCUs

What’s next?

- Check if the resulting readings are correct and if they need noise filtering.
- Adjust sampling rate according to the user’s input.
- Use STM32.AI to deploy model on MCU
- Apply needed preprocessing tasks to captured data and use model for inference