Digital Signal Processing Techniques for Detection and Classification of ECG signals (EEE826 21/22)

Student Names: Jack Gibson, Zeed Alfar

Course Title: MEng (Hons) Mechatronic Engineering + German Masters

Contents

Pan Tompkins Algorithm Introduction

The Pan Tompkins algorithm was initially proposed by Jiapu Pan and Willis J. Tompkins in 1985, their algorithm is highly effective in detection of QRS complexes with a report of 99.3% accurate detection of annotated arrhythmia database. The algorithm can be broken down into two sections, the pre-processing, and the decision rulesets.

The pre-processing is broken down as follows in Figure 1 Pre-processing of ECG signal, the algorithm first sets a series of filters which removes background noise, the derivative step is applied to provide information on the slope of the QRS signal and then the signal is squared to amplify the QRS complex's most dominant peaks.

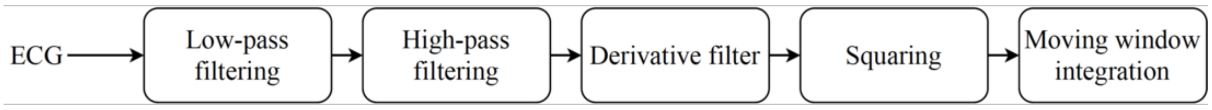


Figure 1 Pre-processing of ECG signal

The decision ruleset stage is where the algorithm incorporates a peak detection strategy using adaptive thresholding to correctly detect peaks within the pre-processed signal. Thresholding is carried out to detect the difference between a signal r-wave peak and a noise peak. Calculations for the thresholds are seen below:

$$THRESHOLD\ I1 = NPKI + 0.25\ (SPKI - NPKI\ )$$

$$ThresholdF1 = NPKF + 0.25(SPKF\ \ - NPKF)$$

To start, a 2-second learning phase is needed first though to initialize SPKI, SPKF, NPKI and NPKF as a percentage of the maximum and average amplitude of the integrated signal and filtered signal. The thresholding is done as the max peak is calculated per window.

$$SPKI = 0.125\ PEAKI + 0.875\ SPKI \qquad \text{if } PEAKI \text{ is the signal peak}$$

$$NPKI = 0.125\ PEAKI\ \ + 0.875\ NPKI \qquad \text{if } PEAKI \text{ is the noise peak}$$

If PEAKI is less than the threshold, NPKI is updated using the equation above and therefore the thresholdI1 is updated. If the PEAKI is higher than the thresholdI1, the same process is carried out for the filtered signal.

$$SPKF = 0.125PEAKF + 0.875SPKF\ \ if\ PEAKF\ is\ signal\ peak$$
$$NPKF = 0.125PEAKF + 0.875NPKF\ \ if\ PEAKF\ is\ noise\ peak$$

If PEAKF is less than threshold F1, NPKF is updated using the equation above and therefore threshold F1 is updated. If PEAKF is higher than thresholdF1, SPKI and SPKF are updated as it means it is an R peak meaning thresholdI1 and thresholdF1 are updated.

The thresholds can be used as the minimum amplitude to look for a peak in both the integrated signal and the filtered signal. It is better to have an adaptive threshold, where it changes after each peak, as opposed to a hard threshold where the threshold is just one value the whole way through the signal. The adaptive method provides a more accurate measurement as the threshold moves to best match its signal.

A check is executed to see if there is a filtered signal peak at every integrated peak location. An additional step can be taken to check for missing QRS complexes after the thresholding, this is in the case of arrhythmia patients where the thresholding may be too high. As the difference to the average sensitivity of the patients from including this extra check is negligible, this method was tested but it may have constituted as a failure as the code could not revert to searching for regular R peaks in one run.

These peaks are then plotted on the integrated signal and a check can be run to see how efficient the algorithm is by applying the annotated peaks from the patient database, checking the locations and amplitudes of the algorithm values against the real values. The Sensitivity and Positive Predictive Value was then found by first finding the true positives, false positives, and false negatives. Using the following formulas, the PPV and Sensitivity can be calculated.

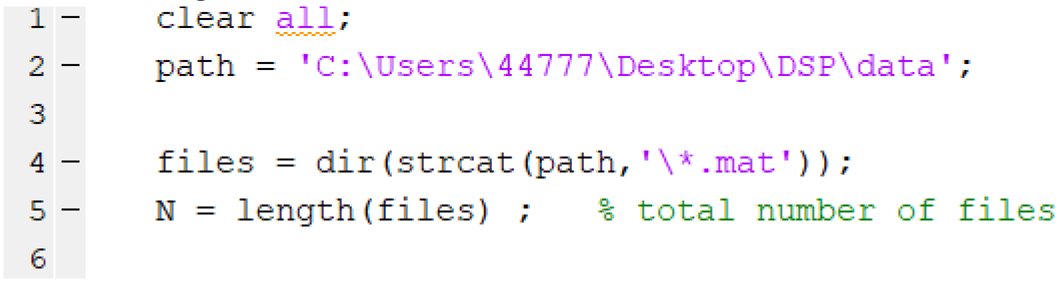
$$\text{Sensitivity} = \frac{\text{True positive}}{\text{True positive} + \text{false negative}} \qquad \text{PPV} = \frac{\text{True positive}}{\text{True positive} + \text{false positive}}$$

Loading in Dataset

The dataset was first required before any of the algorithms could be developed. The code for loading in the dataset in the format of a .mat file has been provided by Dr Pardis Biglarbeigi. The code for this task has been provided in the appendix of this report.

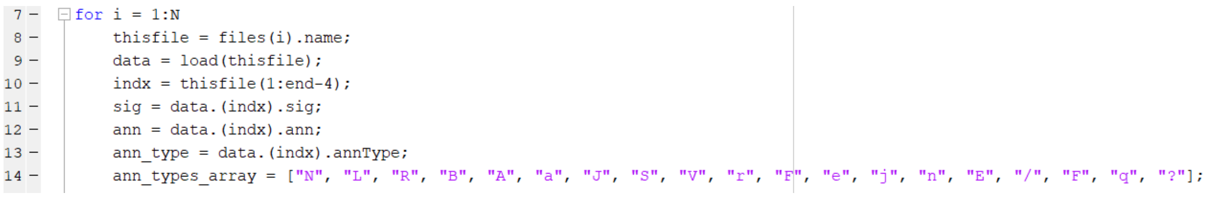
Pan Tompkins MATLAB Implementation

Pre-Processing



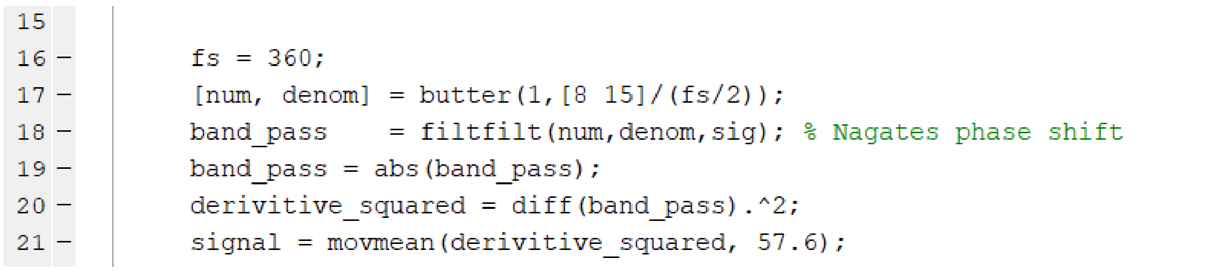
```
clear all;
path = 'C:\Users\44777\Desktop\DSP\data';

files = dir(strcat(path,'\*.mat'));
N = length(files) ;    % total number of files

```

To start the pan Tompkins algorithm, the data is needed from the database. Using a pointer '*' in the directory to call all the .mat files in the specific directory and size of the directory is also taken into consideration.



```
for i = 1:N
    thisfile = files(i).name;
    data = load(thisfile);
    indx = thisfile(1:end-4);
    sig = data.(indx).sig;
    ann = data.(indx).ann;
    ann_type = data.(indx).annType;
    ann_types_array = ["N", "L", "R", "B", "A", "a", "J", "S", "V", "r", "F", "e", "j", "n", "E", "/", "F", "q", "?"];
```

The for loop is to go through each of the 48 mat files within the directory. The files are loaded in one at a time and the signal, peak annotations and peak types are all extracted from the data. It does this by taking the name of each fie and concatenating the name so only the file heading is kept, this is used as an index to call the .sig, .ann and .annType files within the .mat file.



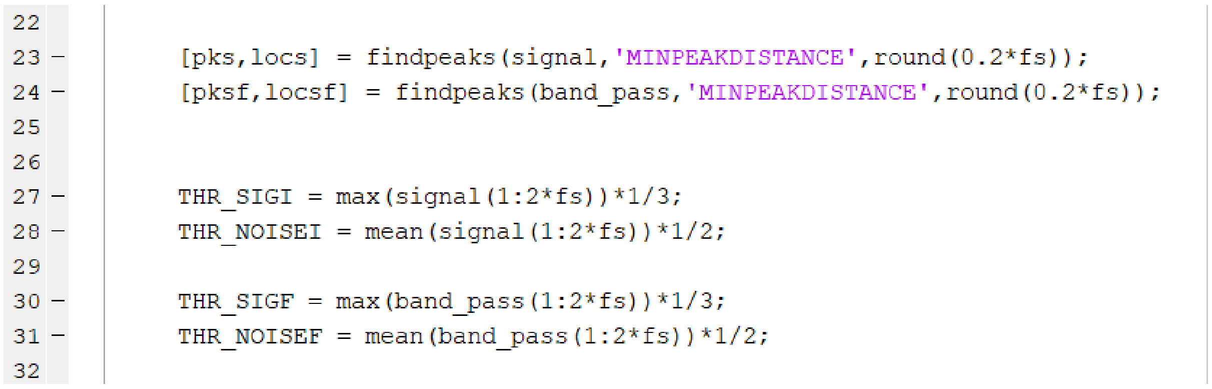
```matlab

    fs = 360;
    [num, denom] = butter(1,[8 15]/(fs/2));
    band_pass    = filtfilt(num,denom,sig); % Nagates phase shift
    band_pass = abs(band_pass);
    derivitive_squared = diff(band_pass).^2;
    signal = movmean(derivitive_squared, 57.6);
```

This part of the algorithm initialises the filtering stage and implements a Butterworth filter with band passing data between 8 and 15Hz. This is because the ECG signal is centred at 10Hz, it was found that by increasing the lower passband, more noise was removed from the signal. This is at the risk of losing valuable information on the ECG data so it is limited to the constrictions of just the lower band as the upper pass band could also have been decreased. The best range was found for detecting the most peaks at 8-15Hz.

By taking the absolute value of the bandpass signal we can improve the performance of the algorithm. This is due to some patients have negativing values for peaks and by taking into consideration just the magnitude and not the phase or direction, more peaks will be found by the algorithm when checking the integrated signal and filtered signal for peaks at the same locations.

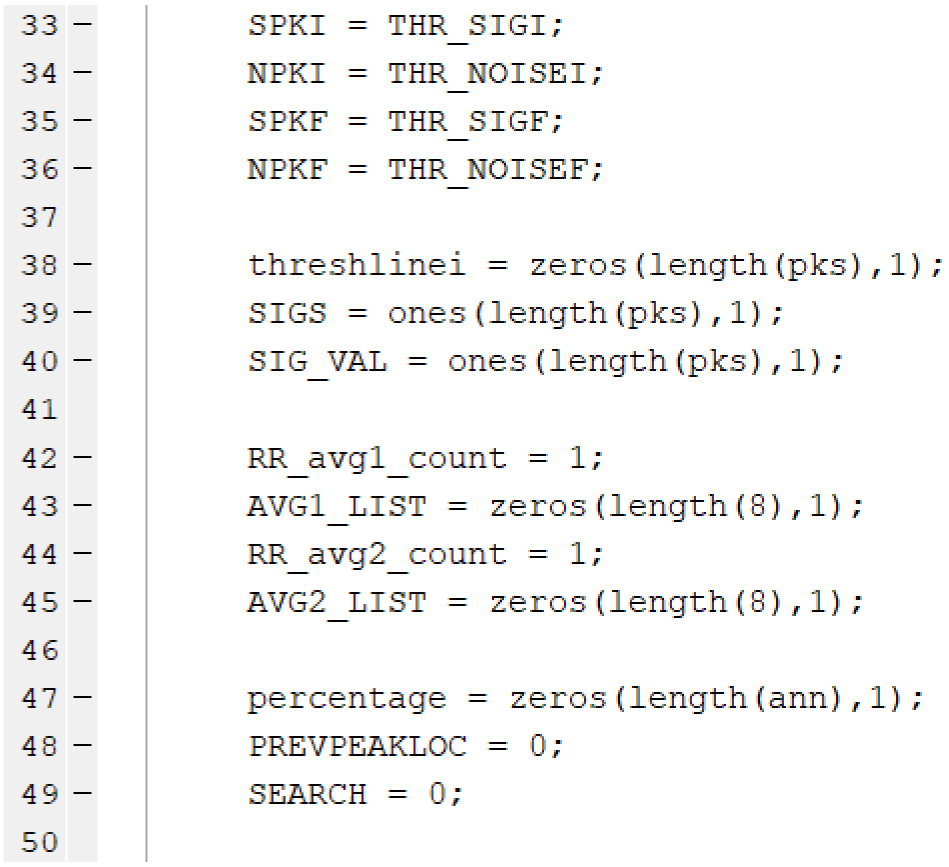
By using the function diff, it takes the first derivative of the filtered signal. The signal is squared after this by using '.^2' and the function movmean, produces a moving average of the squared signal, taking a window of 'peak size' which is the width of a QRS complex in seconds multiplied by the sample frequency.

Thresholding and Decision making



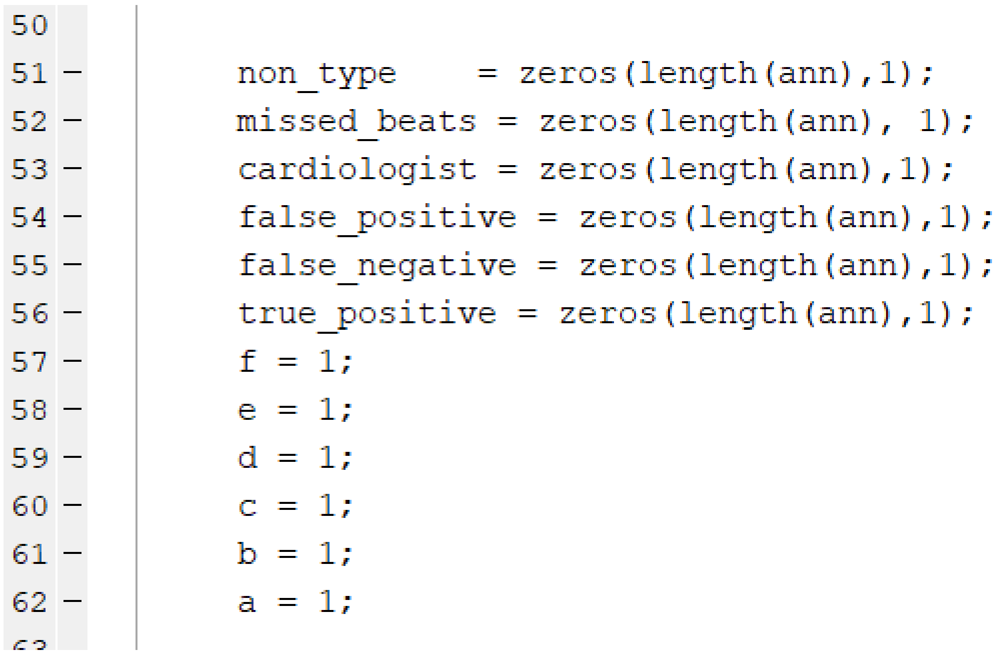
```matlab

    [pks,locs] = findpeaks(signal,'MINPEAKDISTANCE',round(0.2*fs));
    [pksf,locsf] = findpeaks(band_pass,'MINPEAKDISTANCE',round(0.2*fs));


    THR_SIGI = max(signal(1:2*fs))*1/3;
    THR_NOISEI = mean(signal(1:2*fs))*1/2;

    THR_SIGF = max(band_pass(1:2*fs))*1/3;
    THR_NOISEF = mean(band_pass(1:2*fs))*1/2;

```

This next part is for getting the peak locations of the integrated signal and the Butterworth bandpass signal, the rest of the code is the learning phases to initialise the signal level and noise level. Over the first 720 sample points (2 seconds), setting initial NPKI, SPKI, NPKF and SPKF values for the integrated signal and filtered signal, this is called something different as the known variable name sets are used (i.e. NPKI, SPKI, NPKF, SPKF) inside the next FOR loop which will be refined as the code goes on so it is not reinitialised every time inside the loop.



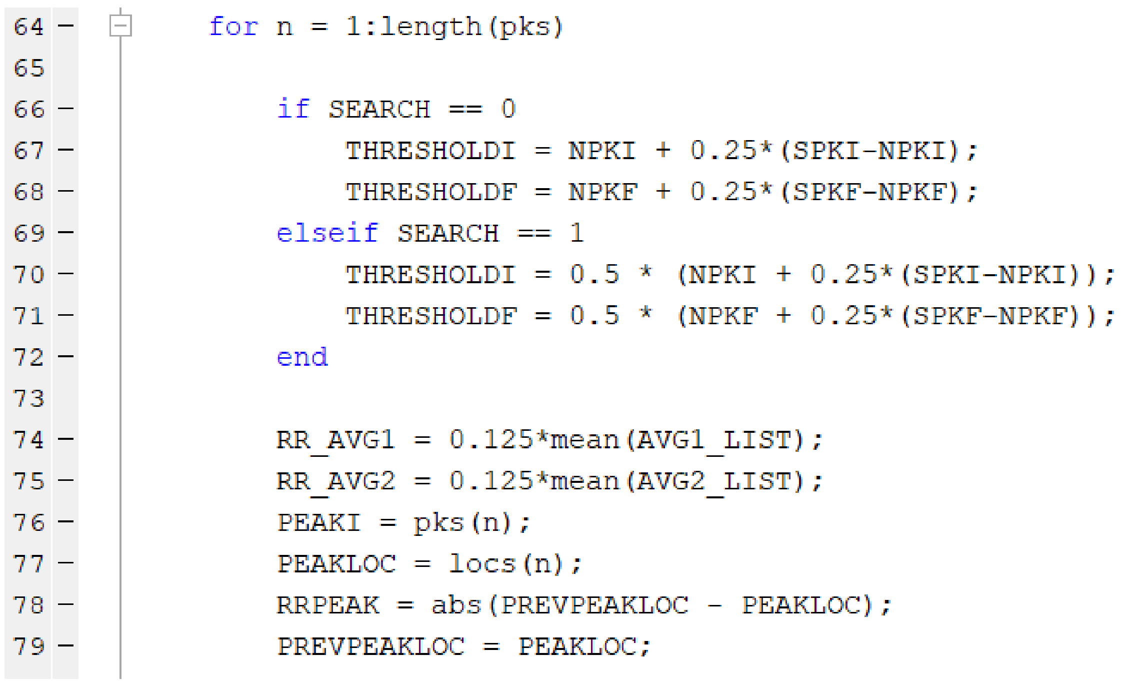
```
    SPKI = THR_SIGI;
    NPKI = THR_NOISEI;
    SPKF = THR_SIGF;
    NPKF = THR_NOISEF;

    threshlinei = zeros(length(pks),1);
    SIGS = ones(length(pks),1);
    SIG_VAL = ones(length(pks),1);

    RR_avg1_count = 1;
    AVG1_LIST = zeros(length(8),1);
    RR_avg2_count = 1;
    AVG2_LIST = zeros(length(8),1);

    percentage = zeros(length(ann),1);
    PREVPEAKLOC = 0;
    SEARCH = 0;

```

This part of the code is initialising all the variables that are to be used in the for loop, the noise and signal levels are renamed to something that is more recognisable with the Pan Tompkins original paper and is easily identifiable and it will be used as the first initial values for the thresholding. The threshlinei is for graphing purposes. The rest of the values are used for the search-back algorithm and calculating when the code should half the threshold for arrhythmia patients.



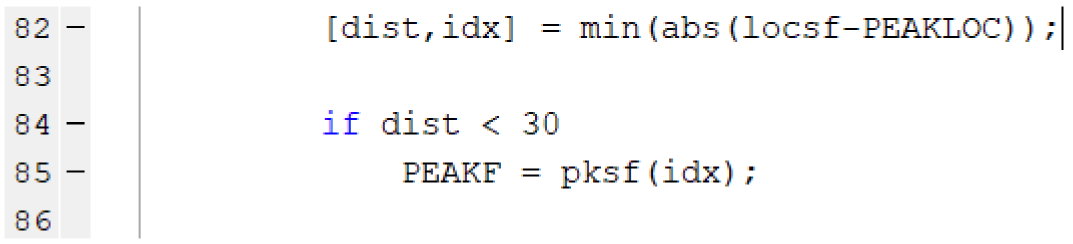
```

    non_type     = zeros(length(ann),1);
    missed_beats = zeros(length(ann), 1);
    cardiologist = zeros(length(ann),1);
    false_positive = zeros(length(ann),1);
    false_negative = zeros(length(ann),1);
    true_positive = zeros(length(ann),1);
    f = 1;
    e = 1;
    d = 1;
    c = 1;
    b = 1;
    a = 1;
```

This part of the code is for the initialising information for the sensitivity and PPV calculations, values for false negatives, false positive and true positives are needed for this calculation.



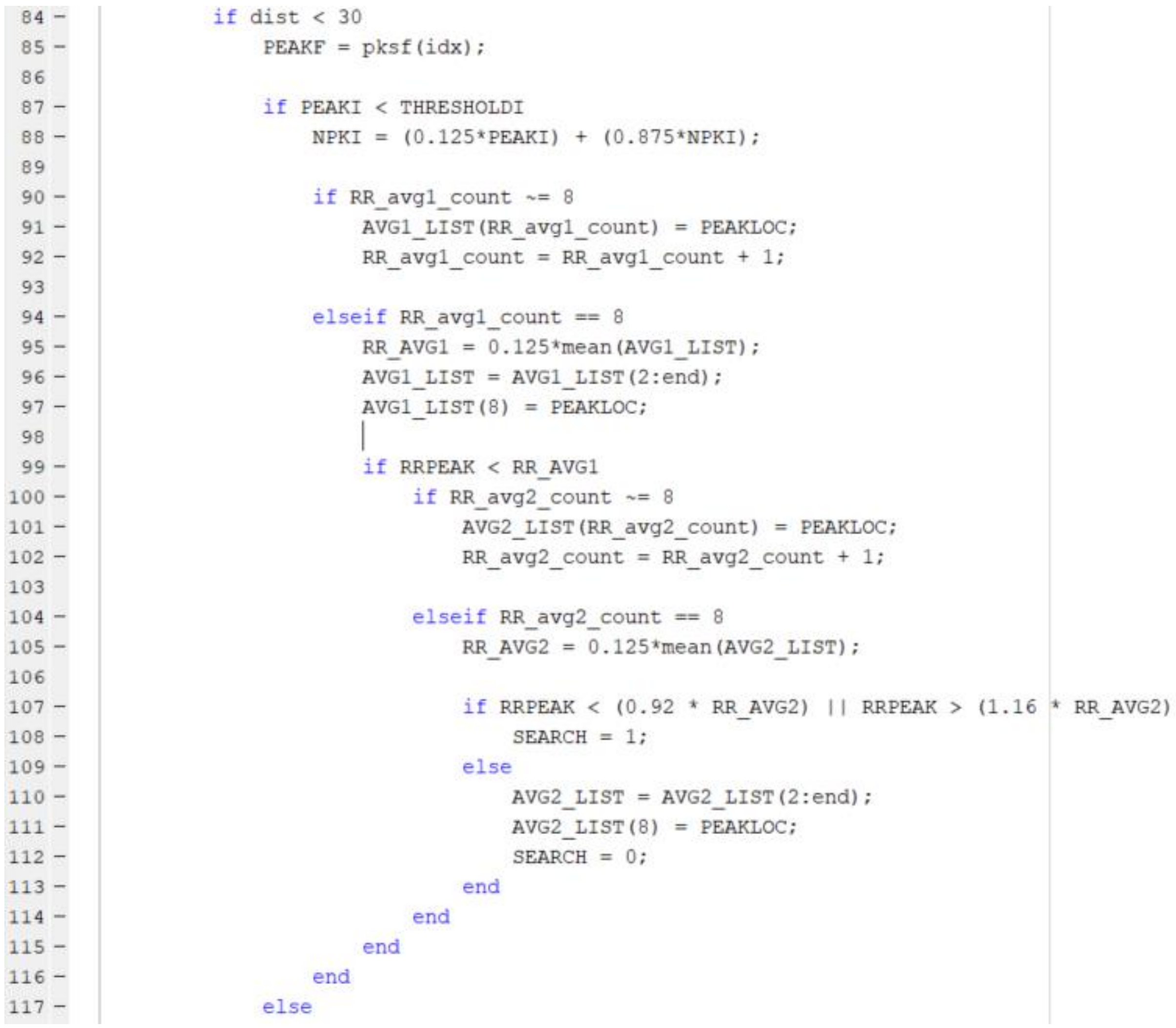
```
    for n = 1:length(pks)

        if SEARCH == 0
            THRESHOLDI = NPKI + 0.25*(SPKI-NPKI);
            THRESHOLDF = NPKF + 0.25*(SPKF-NPKF);
        elseif SEARCH == 1
            THRESHOLDI = 0.5 * (NPKI + 0.25*(SPKI-NPKI));
            THRESHOLDF = 0.5 * (NPKF + 0.25*(SPKF-NPKF));
        end

        RR_AVG1 = 0.125*mean(AVG1_LIST);
        RR_AVG2 = 0.125*mean(AVG2_LIST);
        PEAKI = pks(n);
        PEAKLOC = locs(n);
        RRPEAK = abs(PREVPEAKLOC - PEAKLOC);
        PREVPEAKLOC = PEAKLOC;
```

Start the thresholding, depending on the search back algorithm in place, this is how the thresholding value is selected. If the search back is not triggered, it means the regular thresholds can be uesd, however, if the search back is triggered, the halved thresholds are used in case of arrhythmia patients. PEAKI is then defined as the current peak of the loop and will go through the thresholds to see if it is a true peak of not. RR averages are calculated here as well as RR peak, which is the time period between peaks, this is done by calculating previous peak location and taking the difference between the peak location now.



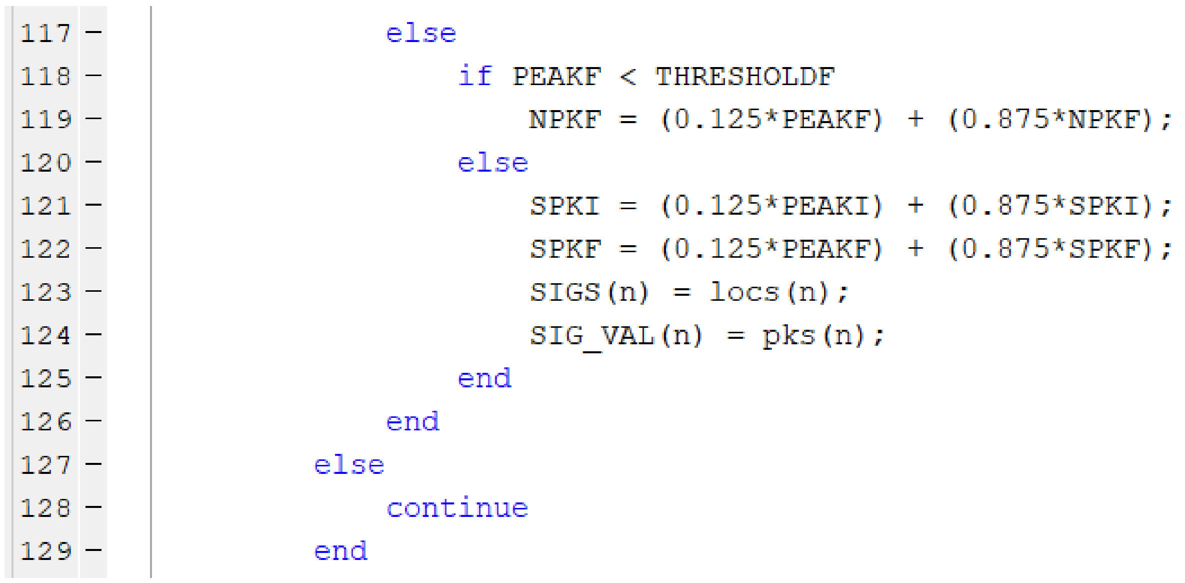
```
        [dist,idx] = min(abs(locsf-PEAKLOC));

        if dist < 30
            PEAKF = pksf(idx);

```

This part of the code is to take corresponding PEAKF only if it is within an accepted tolerance away in peak location from the PEAKI. This is to make sure that PEAKI is also a peak in the filtered signal.



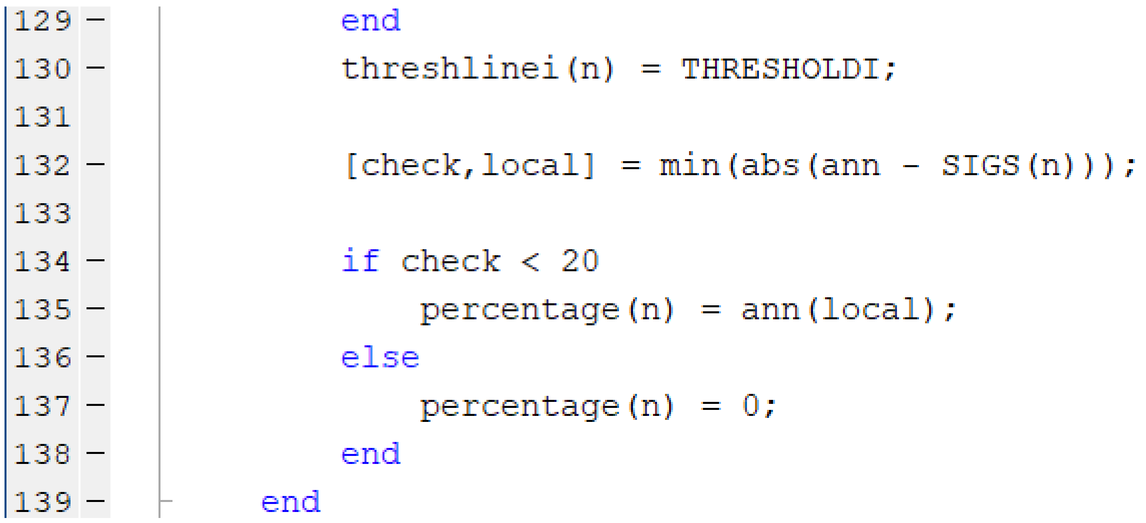
```
 84 -          if dist < 30
 85 -              PEAKF = pksf(idx);
 86
 87 -              if PEAKI < THRESHOLDI
 88 -                  NPKI = (0.125*PEAKI) + (0.875*NPKI);
 89
 90 -                  if RR_avg1_count ~= 8
 91 -                      AVG1_LIST(RR_avg1_count) = PEAKLOC;
 92 -                      RR_avg1_count = RR_avg1_count + 1;
 93
 94 -                  elseif RR_avg1_count == 8
 95 -                      RR_AVG1 = 0.125*mean(AVG1_LIST);
 96 -                      AVG1_LIST = AVG1_LIST(2:end);
 97 -                      AVG1_LIST(8) = PEAKLOC;
 98
 99 -                      if RRPEAK < RR_AVG1
100 -                          if RR_avg2_count ~= 8
101 -                              AVG2_LIST(RR_avg2_count) = PEAKLOC;
102 -                              RR_avg2_count = RR_avg2_count + 1;
103
104 -                          elseif RR_avg2_count == 8
105 -                              RR_AVG2 = 0.125*mean(AVG2_LIST);
106
107 -                              if RRPEAK < (0.92 * RR_AVG2) || RRPEAK > (1.16 * RR_AVG2)
108 -                                  SEARCH = 1;
109 -                              else
110 -                                  AVG2_LIST = AVG2_LIST(2:end);
111 -                                  AVG2_LIST(8) = PEAKLOC;
112 -                                  SEARCH = 0;
113 -                              end
114 -                          end
115 -                      end
116 -                  end
117 -              else
```

From line 87 onwards is the first implementation of the thresholding algorithm, it checks if the peak that was located is below the threshold and if it is then we will update the NPKI value. After this, the RR-average algorithm is then checked. Th initial part of the RR-average decision branches are to fill the array of 8 beats, RR average1 is just the 8 most recent R-peaks while RR-average2 is 8 beats which fall into the limits. The RR-average2 are first filled with R-peaks that go under the RR-average1 and then gets updated to fill with beats that fall within the limit. If the RR-peaks are too far out of limits, then a search back is triggered to lower the threshold.



```
            else
                if PEAKF < THRESHOLDF
                    NPKF = (0.125*PEAKF) + (0.875*NPKF);
                else
                    SPKI = (0.125*PEAKI) + (0.875*SPKI);
                    SPKF = (0.125*PEAKF) + (0.875*SPKF);
                    SIGS(n) = locs(n);
                    SIG_VAL(n) = pks(n);
                end
            end
        else
            continue
        end
```

This part of the code checks if the PEAKI is a peak by checking that its corresponding PEAKF is above of below the threshold. If the PEAKF is below the threshold then NPKF is updated so that the threshold can update for the future use. If it is above the threshold, then it means this is a definite peak and both SPKI & SPKF are updated, the peak's location is then noted and set it in an array for future use.

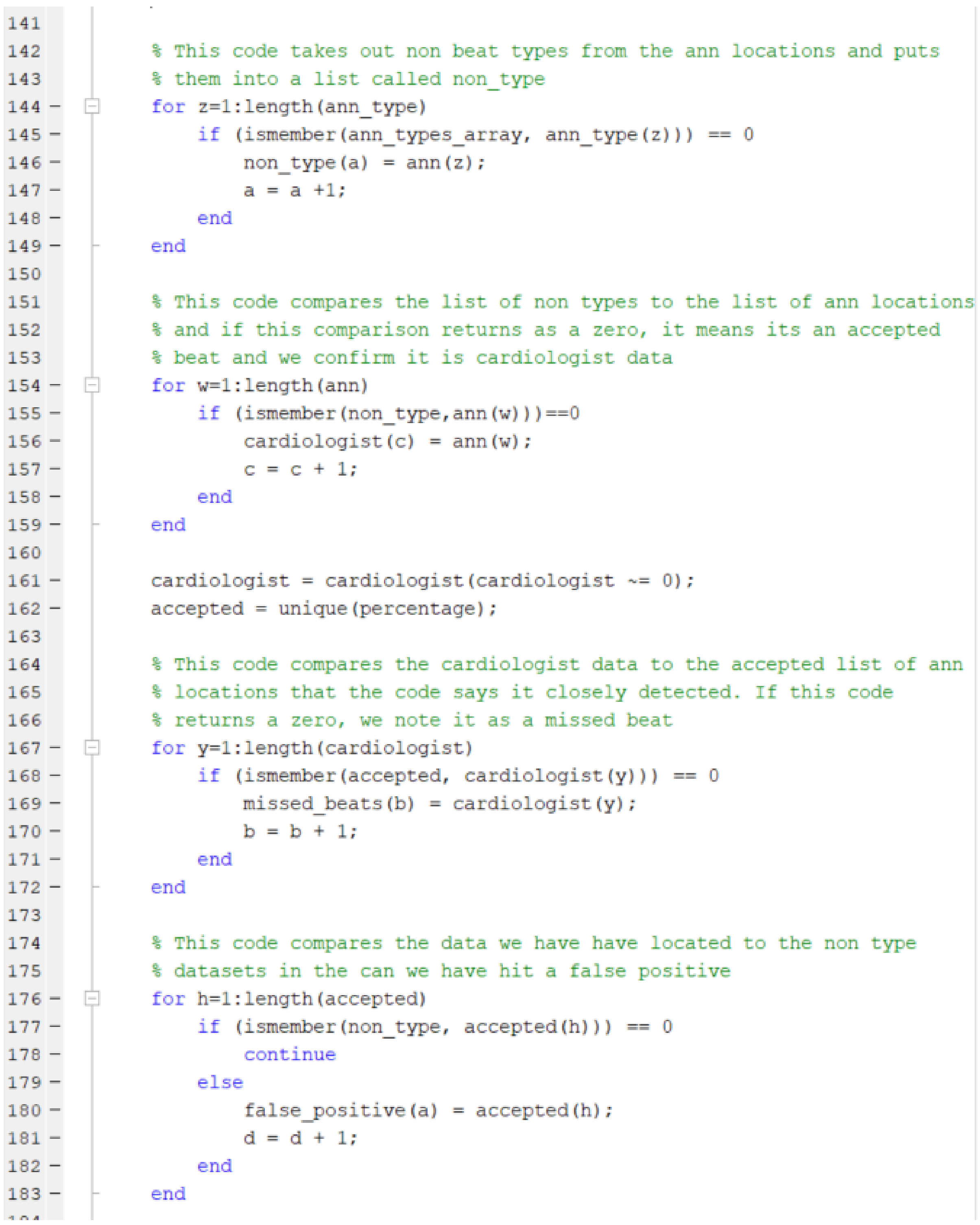


```
        end
        threshlinei(n) = THRESHOLDI;

        [check,local] = min(abs(ann - SIGS(n)));

        if check < 20
            percentage(n) = ann(local);
        else
            percentage(n) = 0;
        end
    end
```

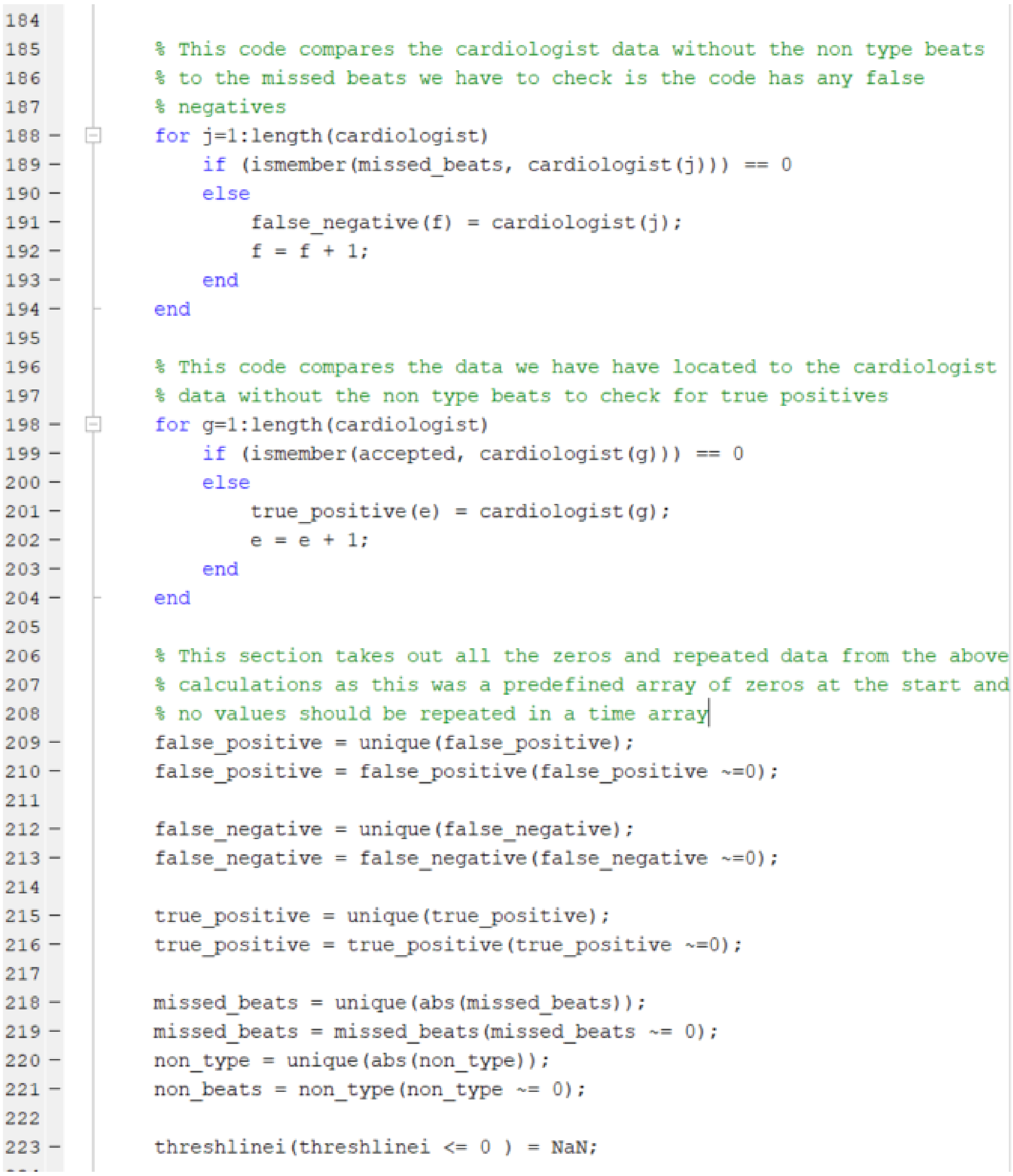
The threslinei is used for graphing purposes and is updated in every iteration here. The next section of the code checks if the peak that was located by our code is within an accepted tolerance to the peak annotated by the cardiologist's data sets, this is for the sensitivity and PPV purposes.

Performance Analysis

The next part of the code is collecting necessary data to be able to calculate false positives, false negatives and true positives as well as using this data to calculate the latter.

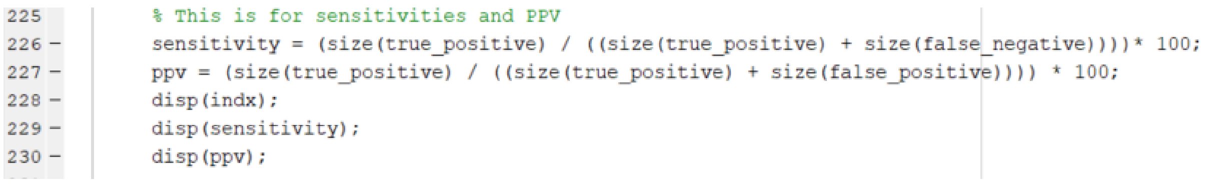


```matlab

% This code takes out non beat types from the ann locations and puts
% them into a list called non_type
for z=1:length(ann_type)
    if (ismember(ann_types_array, ann_type(z))) == 0
        non_type(a) = ann(z);
        a = a +1;
    end
end

% This code compares the list of non types to the list of ann locations
% and if this comparison returns as a zero, it means its an accepted
% beat and we confirm it is cardiologist data
for w=1:length(ann)
    if (ismember(non_type,ann(w)))==0
        cardiologist(c) = ann(w);
        c = c + 1;
    end
end

cardiologist = cardiologist(cardiologist ~= 0);
accepted = unique(percentage);

% This code compares the cardiologist data to the accepted list of ann
% locations that the code says it closely detected. If this code
% returns a zero, we note it as a missed beat
for y=1:length(cardiologist)
    if (ismember(accepted, cardiologist(y))) == 0
        missed_beats(b) = cardiologist(y);
        b = b + 1;
    end
end

% This code compares the data we have have located to the non type
% datasets in the can we have hit a false positive
for h=1:length(accepted)
    if (ismember(non_type, accepted(h))) == 0
        continue
    else
        false_positive(a) = accepted(h);
        d = d + 1;
    end
end
```



```
    % This code compares the cardiologist data without the non type beats
    % to the missed beats we have to check is the code has any false
    % negatives
    for j=1:length(cardiologist)
        if (ismember(missed_beats, cardiologist(j))) == 0
        else
            false_negative(f) = cardiologist(j);
            f = f + 1;
        end
    end

    % This code compares the data we have have located to the cardiologist
    % data without the non type beats to check for true positives
    for g=1:length(cardiologist)
        if (ismember(accepted, cardiologist(g))) == 0
        else
            true_positive(e) = cardiologist(g);
            e = e + 1;
        end
    end

    % This section takes out all the zeros and repeated data from the above
    % calculations as this was a predefined array of zeros at the start and
    % no values should be repeated in a time array
    false_positive = unique(false_positive);
    false_positive = false_positive(false_positive ~=0);

    false_negative = unique(false_negative);
    false_negative = false_negative(false_negative ~=0);

    true_positive = unique(true_positive);
    true_positive = true_positive(true_positive ~=0);

    missed_beats = unique(abs(missed_beats));
    missed_beats = missed_beats(missed_beats ~= 0);
    non_type = unique(abs(non_type));
    non_beats = non_type(non_type ~= 0);

    threshlinei(threshlinei <= 0 ) = NaN;
```

The calculated data sets are then cleaned of zeros from their initialisation as an array of zeros so that they can be used for calculation.



```
    % This is for sensitivities and PPV
    sensitivity = (size(true_positive) / ((size(true_positive) + size(false_negative))))* 100;
    ppv = (size(true_positive) / ((size(true_positive) + size(false_positive)))) * 100;
    disp(indx);
    disp(sensitivity);
    disp(ppv);
```

If the sensitivities of all these datasets are then put into list and the mean averages are calculated, the sensitivity is 99.4295 % and the PPV is 99.9780 %. This is with a tolerance of 20 samples either side of the peaks. The performance tables for 5 samples vs 20 samples are in the appendix.

The Wavelet Transform (WT)

The Wavelet Transform (WT) is a highly effective mathematical function that detects changes in signal frequency in the time domain, that is a wave like oscillation localised in space (time). Many mother wavelets exist, such as Haar, Daubechies Symlets, Coiflets, Biorthogonal and Discrete Meyer, each with its own array of wavelet type and slight variation in shape. Through shifting (location) and scaling (stretching and compressing), the mother wavelet can fit to the components of interest in the signal. The process allows detection of notable points in the signal in both time and frequency domains. A mother wavelet is selected by pairing its shape to the approximate shape of frequency changes in the signal it is to identify. This critical step matches the most appropriate mother wavelet to fit the signal components to be analysed.

The Wavelet Transform (WT) is defined as:

$$W_i\, x(k) = \frac{1}{\sqrt{i}} \int_{-\infty}^{\infty} x(t)\, \Psi\left(\frac{t-k}{i}\right) dt$$

Where "i" and "k" represent the dilation and translation parameters, respectively.

The basic concept of wavelet transform is to select a scale of wavelet and drag it along the signal iterating through each of the samples in the signal. Each point in the signal is multiplied with the wavelet resulting in convolution between the wavelet and the signal. Once completed for one scale, the scale is changed, and the process of convolution is repeated for the new scaled wavelet.

The Wavelet Transform (WT) includes Discrete Wavelet Transform (DWT) and Continuous Wavelet Transform (CWT).

This report details how these two mathematical functions are used to create algorithms for the analysis of ECG data by

1. Discrete Wavelet Transform (DWT) which is used to **detect** QRS complexes in the MIT-BIH Database
2. Continuous Wavelet Transform (CWT) which is used to train a Convolutional Neural Network (CNN) to **classify** ECG data from three different datasets: patients with Arrhythmia (ARR), Congestive Heart Failure (CHF) and Normal Sinus Rhythm (NSR)

Discrete Wavelet Transform (DWT) Concept

The Discrete Wavelet Transform (DWT) is rapidly becoming one of the most advanced signal processing algorithms. It allows for great time resolution at low frequencies as well as great frequency resolution at low time resolution. The process of a wavelet transform is to first do a forward DWT, select coefficients to save, then calculate the inverse DWT to retain only the frequencies useful for analytics. DWT is a process of decomposition of the frequency spectra into discrete frequency bins defined as the coefficients.

Continuous Wavelet Transform (CWT) Concept

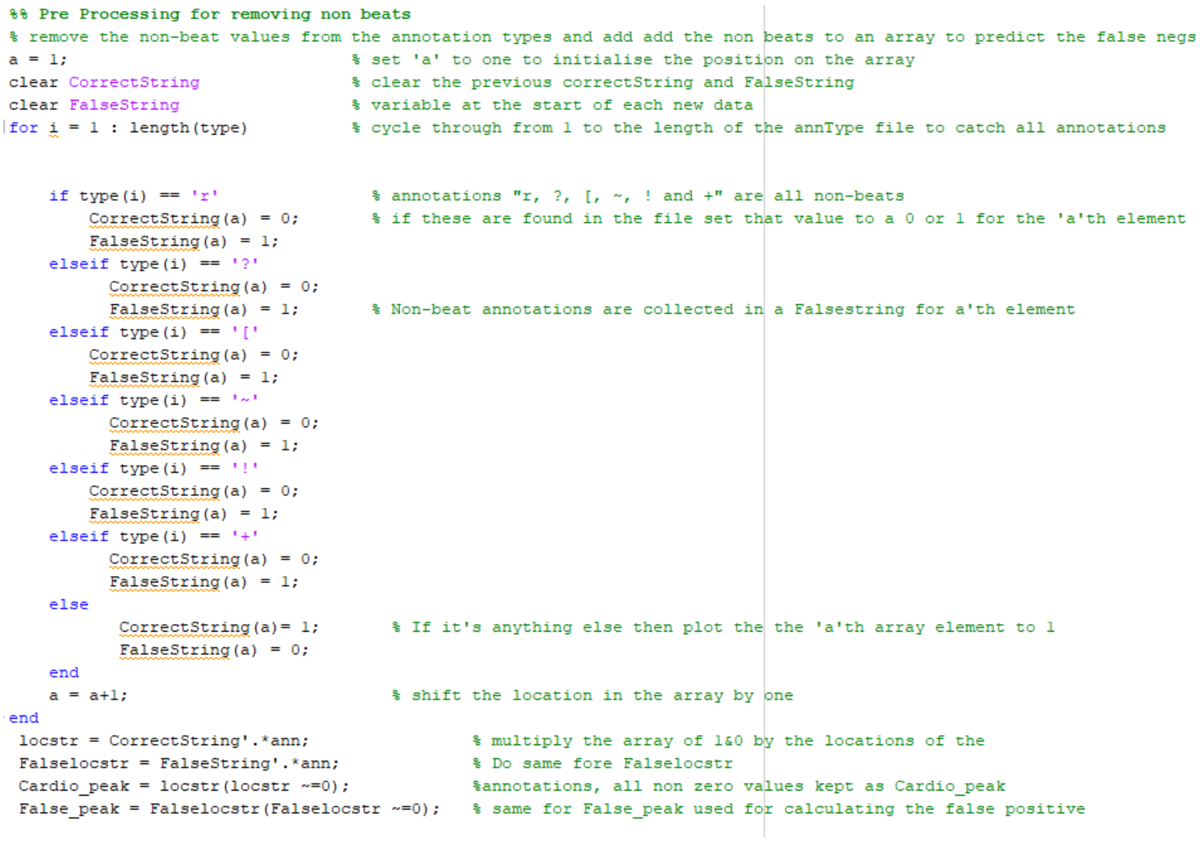
The Continuous Wavelet Transform (CWT) employs a similar process to that of the Discrete Wavelet Transform (DWT). The coefficients are an inner product of a convolution for the similarity between signals and their analysing function (mother wavelet) The signal is compared to the scaled and dilated versions of the wavelet. These coefficients are mapped to a scalogram showing the signal as a function of time and frequency. [6]

Detection of QRS Complexes using Discrete Wavelet Transform (DWT)

Pre-Processing

The initial data loaded into the MATLAB workspace is annotated by cardiologists with markers. These markers are extremely useful for analysing the performance of the algorithm. However, not all annotations made by the cardiologist represent the locations of QRS complexes. The annotations to be removed and their meanings are defined in the appendix. These 'Non-Beat' values are useless for defining QRS complexes, however, they are of vital importance for analysing the performance of the algorithm after its development and so should be stored in an array for later use.

The code used to show how the cardiologists' annotations are split into two separate arrays comprising Beats and Non-Beat values. In reality the list of Non-Beat annotations are longer than shown in the code below, these have been omitted from this segment for readability of this report. However, the full set of non beats have been removed for the use in the calculation of performance later.



```
%% Pre Processing for removing non beats
% remove the non-beat values from the annotation types and add add the non beats to an array to predict the false negs
a = 1;                              % set 'a' to one to initialise the position on the array
clear CorrectString                 % clear the previous correctString and FalseString
clear FalseString                   % variable at the start of each new data
for i = 1 : length(type)            % cycle through from 1 to the length of the annType file to catch all annotations


    if type(i) == 'r'                   % annotations "r, ?, [, ~, ! and +" are all non-beats
        CorrectString(a) = 0;           % if these are found in the file set that value to a 0 or 1 for the 'a'th element
        FalseString(a) = 1;
    elseif type(i) == '?'
          CorrectString(a) = 0;
          FalseString(a) = 1;           % Non-beat annotations are collected in a Falsestring for a'th element
    elseif type(i) == '['
        CorrectString(a) = 0;
        FalseString(a) = 1;
    elseif type(i) == '~'
          CorrectString(a) = 0;
          FalseString(a) = 1;
    elseif type(i) == '!'
        CorrectString(a) = 0;
        FalseString(a) = 1;
    elseif type(i) == '+'
          CorrectString(a) = 0;
          FalseString(a) = 1;
    else
           CorrectString(a)= 1;          % If it's anything else then plot the the 'a'th array element to 1
           FalseString(a) = 0;
    end
    a = a+1;                             % shift the location in the array by one
end
 locstr = CorrectString'.*ann;                   % multiply the array of 1&0 by the locations of the
 Falselocstr = FalseString'.*ann;                % Do same fore Falselocstr
 Cardio_peak = locstr(locstr ~=0);               %annotations, all non zero values kept as Cardio_peak
 False_peak = Falselocstr(Falselocstr ~=0);      % same for False_peak used for calculating the false positive
```

Within the 'For' loop used for cycling through all patients in the dataset there is another 'For' loop with the length of 'type' file (Cardiologist annotations). A series of 'if' statements follow used to show the locations of Beats and Non-Beat values within the 'types' array. If the value within the array is one of 'Non-Beat' status, then a '0' is put in the CorrectString array whilst a '1' is put into the FalseString array. Using the variable 'a' the location within the CorrectString and FalseString arrays can be iterated through as the For loop moves along the 'type' array. This process continues until the whole length of 'types' has been cycled through resulting in two arrays comprising opposite '0' and '1' values.

To remove the Beat or Non-Beat values, the CorrectString and FalseString arrays are multiplied with the 'ann' array. The ann array shows the location of these beats, by multiplying by 0 the Non-Beat location is removed from the string and replaces with a 0 whereas multiplying by 1 keeps the location.

The final two arrays for the locations of Beats and Non-Beats are created by keeping values which are not equal to zero resulting in Cardio_peak for Cardiologist Beat locations and False_peak for locations of the Non-Beat values. For a new array of Cardio_peak and False_peak to be created for each new patient, both arrays are cleared at the start of the code to avoid dimensional errors.

Discrete Wavelet Transform (DWT)

To execute a forward Discrete Wavelet Transform the code below is used. The Maximal Overlap Discrete Wavelet Transform Function (MODWT) is used. The function takes three inputs, that is the signal it is to segment (ecgsig), the number of levels that the function is to break the signal down into (5) and the type of wavelet used in the transform (Symlet 4). Due to the nature of the MODWT the wavelet selected is required to be orthogonal.

```
%Descrete wavelet transform
wt = modwt(ecgsig, 5,'sym4');    % 5 level undecimated DWT using the ...
                                 %'sym4' wavelet as it matches the normal ecg beat best
```

The ECG frequencies have been broken into 5 coefficients, as shown in the table below. It is widely accepted that the most important frequency information in the ECG signal is in the range of 5Hz - 15Hz. For this reason, the frequency coefficients to be kept are d4 and d5, retaining frequencies from 5.625Hz – 22.5Hz.

Coefficient	Frequency Components
d1	90Hz-180Hz
d2	45Hz-90Hz
d3	22.5Hz-45Hz
d4	11.25Hz- 22.5Hz
d5	5.625Hz-11.25Hz

Table 1: Frequency Components of the decomposed ECG signal

To analyse frequency coefficients within d4 and d5 all other frequency bins must be set to zero. The coefficients are then put through an Inverse Discrete Wavelet Transform. This process is shown below:

```
wtrec = zeros(size(wt));          % array of zeros that makes length of wt all zero

wtrec(4:5,:)=wt(4:5,:);           % the d3 and d4 coefficents extracted as ...
                                  % frequencies we want lie in this area

%inverse transform back but with zeros everywhere except where we keep
y1 = imodwt(wtrec, 'sym4');
```

An array wtrec is defined containing zeros the same size as the MODWT array, rows 4 and 5 are replaced with the coefficients from d4 and d5. The imodwt function is used to perform an inverse maximum overlap Discrete Wavelet Transform.

As previously discussed, selection of the most appropriate wavelet is a key factor in successful implementation of the algorithm. Fortunately, there have been many studies into this topic and many authors such as J. Rahul et al [2] and M. Jose Da Silva [3] agree that the “Symlet 4” wavelet is best suited to identifying the QRS complexes in a standard ECG signal.

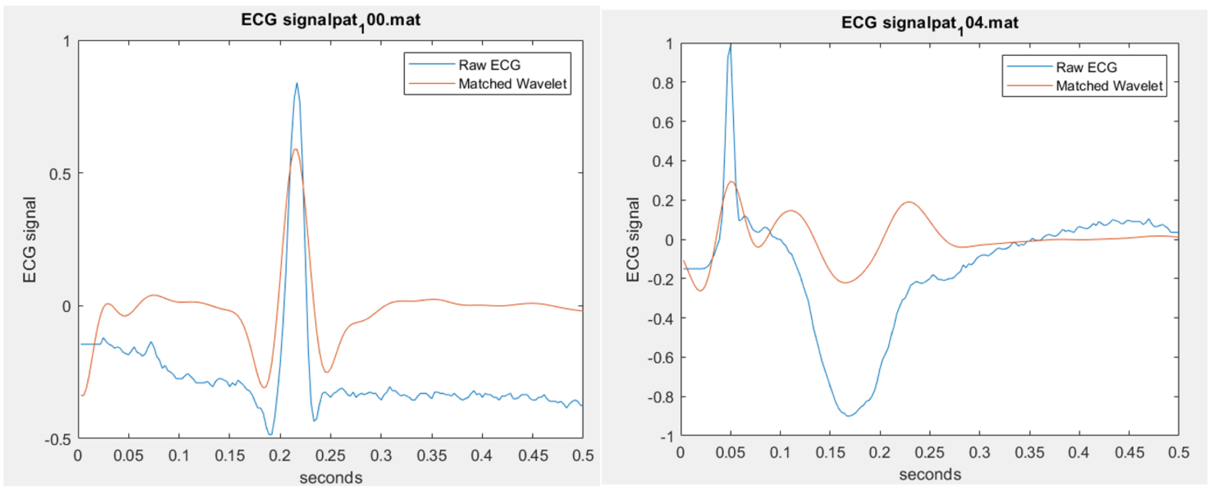


Figure 2: Matching of sym4 wavelet to Normal Sinus Rhythm (left) and sick patient (right)

Although the wavelet Sym4 exhibits the best performance overall when testing on a range of patients, it's performance on some selected very sick patients (such as pat_104, pat_208 and pat_231) suffers. This is because the ECG signals displayed by these sick patients are significantly different from a healthy patient. For this reason, the conventional Sym4 struggles to fit its shape to the shape of the sick patient, as shown in the graphic on the right above (patient 204).

Later in this report the possibility of using a dynamic process to select the most appropriate wavelet for a patient is explored using a Deep Transfer Learning Classification algorithm based on Continuous Wavelet Transform (CWT)

Thresholding

The next stage in the process is setting the threshold for the detection of the QRS complexes. The wavelet will match itself to the shape of the QRS complex in the ECG and the highest peak of the wavelet should be the R wave.

Thresholding is an important step as it allows other lower peaks of the wavelet to be suppressed whilst allowing the larger R wave to be identified correctly. Further processing of the wavelet is necessary before selecting the peaks of the signal, shown below:

```
%% Set threshold for Peak Prediction
y = abs(y1).^2;                   % square the magnitude to make all +ve

Threshold = mean(y);              % threshold moves with each new ecg signal
```

The inversed wavelet transform with redundant frequencies removed is first made absolute and squared. This ensures that all values are made positive. In real world applications it is possible that due to human error cardiologists incorrectly placed ECG leads, mixing up the left and right arm nodes. In this case the R wave will be inverted and will spike downwards instead of up. By taking its absolute value and squaring this otherwise harmful effect can be avoided. Additionally, if a patient is sick with cardiac problems such as Dextrocardia, their R wave may be inverted naturally and so the absolute value must be taken and squared to make positive and exaggerate the signal.

In many implementations of a wavelet transform for R peak detection such as MATLAB [4] a hard threshold is set for all patients. Although this method works to an appropriate level, the performance of the algorithm can be improved by taking the average of the signal of each patient and using that for thresholding.

This thresholding is used in the function findpeaks for finding the peaks in the wavelet signal. The snippet of code that is used for finding the peaks in the absolute squared signal and for calculating the beats per minute is shown below.

```
Fs = 360;                       % set the sampling frequency
samples = 1:length(ecgsig);     % samples set to length of the ecg signal
tx = samples./Fs;               % timing vector counts in seconds
[Rpeaks,Calculated_locs] = findpeaks(y,samples,'MinPeakHeight', Threshold,'MinPeakDistance',54);

beat_num = length(Calculated_locs);                 % No.of beats = No.ofPeaks
timelimit = length(ecgsig)/Fs;                      % time limit in secs
BPM(b) = (beat_num*60)/timelimit;                      % calculate the beats per min
```

The Sampling frequency is defined as 360 Hz, the samples in the signal are defined by the length of ECG signal. Tx is defined as the timing vector which counts from zero to the number of seconds the ECG signal runs for. The find peaks function creates 2 arrays, Rpeaks that show the magnitude of the R wave detected and Calculated_locs shows the location in the sample domain of the R peak.

The lower segment of code shows the calculations for the heart rate of the patient. Firstly, the number of beats in the signal is calculated. Next the time the ECG signal runs for is calculated and then the beats per min is calculated using these two values.

After the QRS peak has been detected the data is displayed in a figure for ease of understanding. The code below shows how a simple subplot has been used to display the data.

```
figure;
subplot(211);
plot(tx, ecgsig);               % plot the raw ecg data for 10 secs
xlim([0,10]);
hold on
plot(tx, y1);                   % Plot the matched wavelet over the top of the ecg
grid on;
xlabel('seconds');
ylabel('ECG signal');
title(strcat('ECG signal   ',num2str(currentfile)));
legend('Raw ECG','Matched Wavelet');
```

The top window of the subplot displays the raw ECG data in seconds. The length of the display is limited to 10 seconds. The matched wavelet signal is then superimposed on the ECG signal.

```
subplot(212)
plot(samples,y);              % plot the abs. squared wavelet signal against the samples
grid on
xlim([0,3600]);               % limit to 10 sec of samples (@360Hz)
hold on
plot(Calculated_locs, Rpeaks, 'ro');     % plot the predicted R peaks on top of the wavelet sig
hold on
plot(Cardio_peak, y(Cardio_peak), '*')   % plot the cardiologist values on top
legend('Preprocessed Sig','Predicted "R" Peak', 'Cardiologist "R" Peak', 'Location','southwest');
xlabel('Samples');
title(strcat('R Peaks Found and Heart Rate:  ' , num2str(BPM(b))));
```

The lower window of the subplot shows the absolute squared wavelet signal with the predicted R wave at the peak shown as a red circle. The cardiologist's annotations for R waves are plotted as yellow stars to visually assess accuracy of the predictions. The heart rate (BPM) of each patient is displayed in the title. The results for patient 100 are shown below.

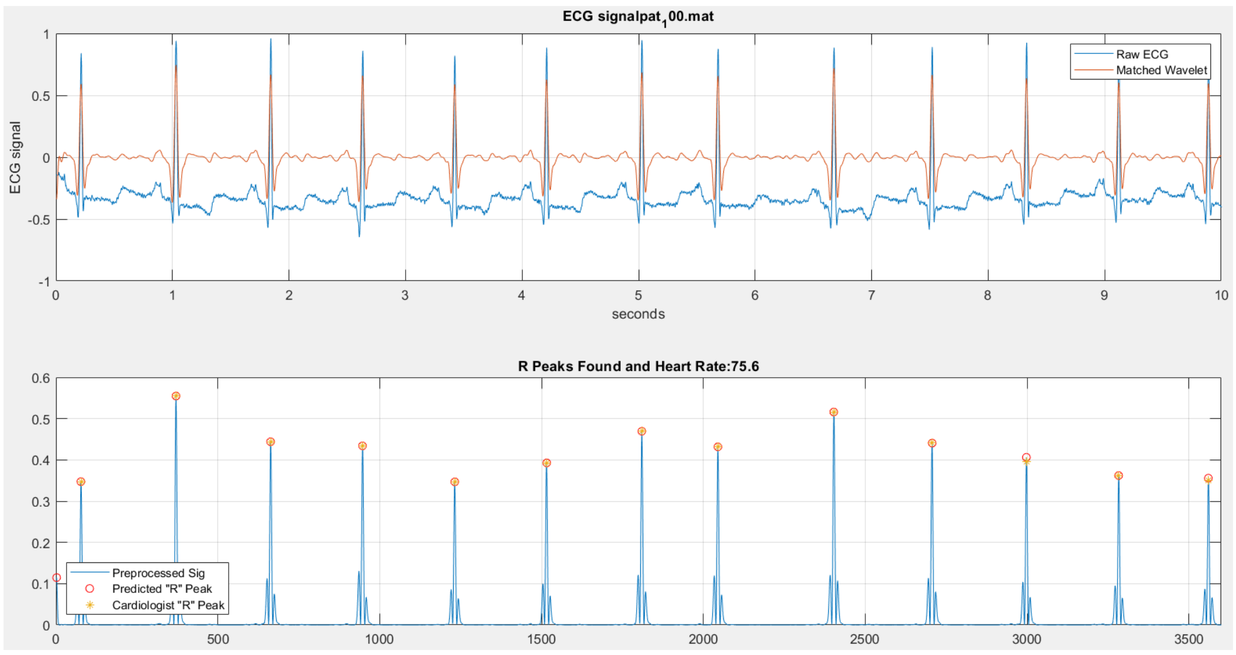
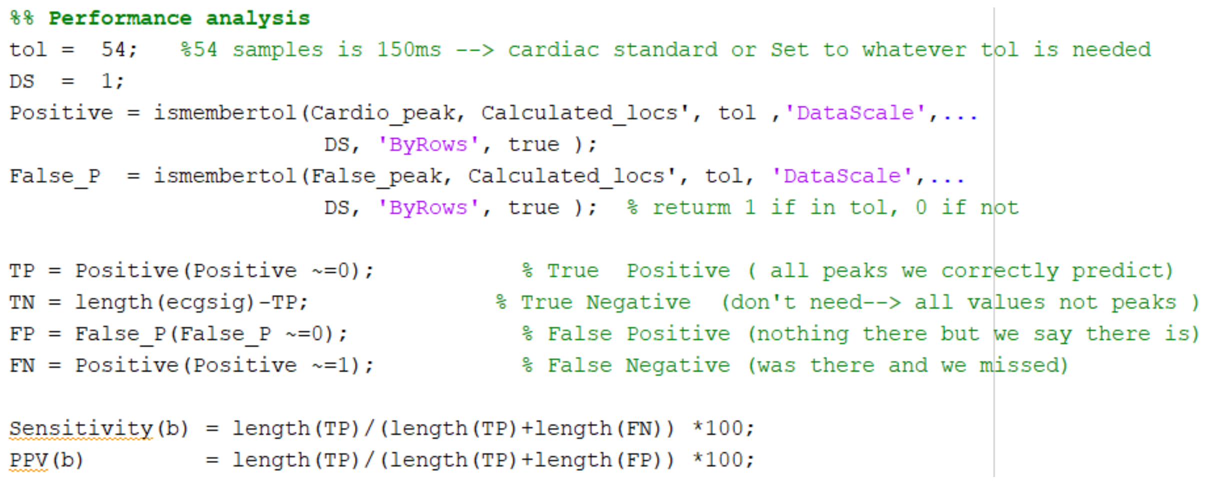


Figure 3: Plot showing raw ECG with matched wavelet (Top). Absolute Squared wavelet with Predicted and Cardiologist R Peaks (Bottom)

Performance Analysis

Algorithm performance is analysed using two methods: Sensitivity and Positive Predictive Value (PPV). Another method of analysing the performance of algorithms is Specificity, however as the number of peaks compared to no peaks is so small the result of this test would always be greater than 99.99% and so analysis by Specificity is not included in this report.

The method for analysis is shown below:



```
%% Performance analysis
tol =  54;    %54 samples is 150ms --> cardiac standard or Set to whatever tol is needed
DS  =  1;
Positive = ismembertol(Cardio_peak, Calculated_locs', tol ,'DataScale',...
                        DS, 'ByRows', true );
False_P  = ismembertol(False_peak, Calculated_locs', tol, 'DataScale',...
                        DS, 'ByRows', true );  % returm 1 if in tol, 0 if not

TP = Positive(Positive ~=0);           % True  Positive ( all peaks we correctly predict)
TN = length(ecgsig)-TP;               % True Negative  (don't need--> all values not peaks )
FP = False_P(False_P ~=0);             % False Positive (nothing there but we say there is)
FN = Positive(Positive ~=1);           % False Negative (was there and we missed)

Sensitivity(b)  = length(TP)/(length(TP)+length(FN)) *100;
PPV(b)          = length(TP)/(length(TP)+length(FP)) *100;
```

The tolerance of a positive prediction is defined, as the algorithm is unlikely to get the prediction of the R wave correct every time in comparison with cardiologists' values (less than 0.00138 secs). Different tolerances have been set to show the performance of the algorithm at different accuracies. The tolerances under test are 54 samples as defined by the NCBI standard for the prediction of R peaks [5]. In addition, tolerances of 5 samples are used to show how the algorithm behaves in high degree of accuracy, the PPV is unrealistically high mostly reaching 100%. In comparison, a tolerance of 500 samples shows how if the window of acceptance is increased Sensitivity values become unrealistically high. A full table of each of these tolerance performances can be found in the appendix under Table A, Table B and Table C.

The function ismembertol is used to compare if the values predicted by the algorithm are within the tolerance set by the user. The function creates an array of 1s and 0s depending on if the prediction is within tolerance.

The array Positive is defined to calculate the positive predictions of the R peaks. Within the function the array Calculated_locs is compared with the array Cardio_peak discussed in the Pre-processing section. The tolerance is defined and scaled to the data using the DS and Data Scale function variables. The function is instructed to look at both arrays by their rows and to return a 1 if each of the numbers is within tolerance.

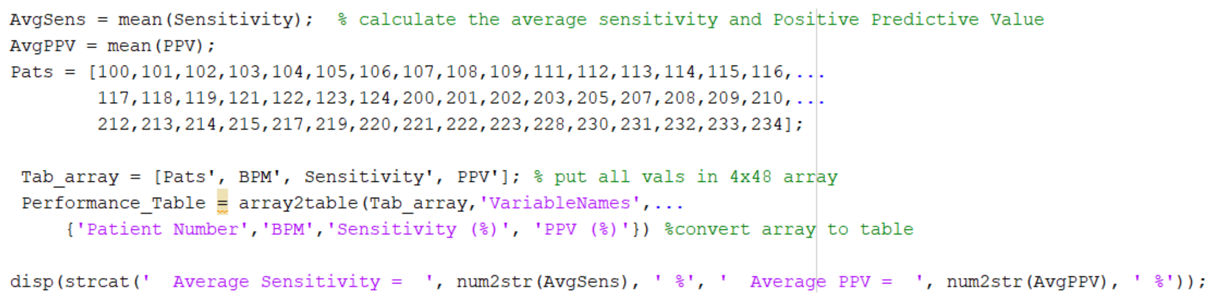
The same process is used for defining the array False_P however the Cardio_peak array is swapped out for the False_peak array. This calculates if the algorithm has falsely predicted any of the Non-Beat values annotated by the cardiologist as QRS complexes.

The True Positive values are defined as an array TP and is created by removing all zeros (not predicted) values from the Positive array. TN is defined as True Negative and is not required for the calculations of Sensitivity and PPV. It is the total length of the ECG signal with the number of QRS complexes removed. FP is defined as an array for the false positive values, the values withing the False_P array that are not zeros are kept leaving only the false QRS predictions. If the algorithm works as intended the number of false positives should be as low as possible. False Negatives are defined within the array FN and are all the peaks that the algorithm should have detected but failed to, this is all values in the Positive array that are not 1 (everything not predicted).

Sensitivity and PPV equations are shown on page 4 of this document in the Pan Tompkins introduction.

These same equations are replicated in the MATLAB script and are converted to a percentage.

Once all patients have been run through the algorithm in the loop the results for each are compiled into a table and the average Sensitivity and PPV is calculated. The code for this step is shown below:



```
AvgSens = mean(Sensitivity);  % calculate the average sensitivity and Positive Predictive Value
AvgPPV = mean(PPV);
Pats = [100,101,102,103,104,105,106,107,108,109,111,112,113,114,115,116,...
        117,118,119,121,122,123,124,200,201,202,203,205,207,208,209,210,...
        212,213,214,215,217,219,220,221,222,223,228,230,231,232,233,234];

 Tab_array = [Pats', BPM', Sensitivity', PPV']; % put all vals in 4x48 array
 Performance_Table = array2table(Tab_array,'VariableNames',...
     {'Patient Number','BPM','Sensitivity (%)', 'PPV (%)'}) %convert array to table

disp(strcat('  Average Sensitivity =  ', num2str(AvgSens), ' %', '  Average PPV =  ', num2str(AvgPPV), ' %'));
```

The average Sensitivity and PPV are both calculated simply using the mean function and displayed at the end of the algorithm running. Values for each patient such as Heart Rate, Sensitivity% and PPV% are printed into a table along with their Patient number. This is done by first creating an array with the Pat. Number, BPM, Sensitivity and PPV. Using the array2table function the contents of the array are then assigned their variable names and output in the command window and stored as a table. The complete output table for all patients can be found in the appendix. The final resulting Sensitivity of the algorithm was found to be 99.3% and an average PPV of 99.4% for a tolerance of 54 samples (150ms).

Deep Learning Method for Classification of ECG Data Using Continuous Wavelet Transform (CWT)

The main cause for a lack of Sensitivity or Positive Predictive Value (PPV) within the Maximal Overlap Discrete Wavelet Transform is that the specified wavelet does not fit accurately to the cardiac signal of the patient. A Deep Transfer Learning method for ECG classification was used to classify Arrhythmia (ARR), Congestive Heart Failure (CHF) and Normal Sinus Rhythm (NSR). All code files are available in the accompanying zip file.

Pre-processing

As classification occurs via a process of Transfer Learning, the precompiled Convolutional Neural Network (CNN) selected to be used is Alex NET. Initially, raw ECG signals are converted to images to allow training and testing. Data is loaded, with 80% of patient data from each class, (ARR, CHF, NSR) separated into a training set and the remaining 20% retained to be used for a test set.

The filter bank is created using the cwtfilterbank() function to define signal length, type of wavelet used (Analytic Morlet) and the voices/bandpass filters per octave (12). This filter bank is then used as an input to the defined function ecg2cwtscg and ecg2cwtscgTEST. Both functions have been created to convert the signal to a spectrogram of n samples. Both functions are identical in process differing only in location where data is stored and size of the loops. The test data is stored to the Test Directory and the training data to the Training Directory.

Within these functions the ecgtype is read (ARR, CHF or NSR) in an if statement. The first ECG signal is taken in and split into sections of a defined number of samples. This first snippet's Continuous Wavelet Transform (CWT) coefficients are taken and normalised between zero and one. It is then mapped to a spectrogram and resized into an Alexnet compatible format (227x227). The image is then stored in its corresponding directory and the process repeated for the further snippets left in the signal and the rest of the ECG signals.

Transfer Learning

To effectively use the CNN for Transfer Learning most of the precompiled model is retained with corresponding weights and biases. The last three layers however are to be reconfigured for the development of the new trained model. The fully connected layer, soft max layer and the classification layer have been altered to allow for the CNN to classify the types of ECG. The Weight Learning Rate Factor has been set to 20 along with the Bias Learning Rate Factor that has also been set to 20.

The training options were then defined, the stochastic gradient descent with momentum algorithm was used for updating training parameters, Maximum epoch was set to 5 as the model generally did not change accuracy significantly past 5 epochs. The learning rate of 0.0001 was used, all data was set to be shuffled after each epoch. The two directories of training and testing data are referenced and included to train the model and validate its performance. The accuracy of the model is calculated, and a confusion matrix plotted.

Optimisation

To achieve the best model accuracy different parameters were changed in the training of the model. Firstly, the number of snippets taken from the signal was changes along with the signal length of each image, increasing the length of the signal for the image reduces the impeding effect of the cone of influence on the signal. Firstly, the images used were 500 samples long with 10 images per patient (Figure A in Appendix). However, the cone of influence was too steep due to the limiting number of samples. The Number of samples were increased to 1000 with 65 images per patient (Figure B in Appendix). This improved the performance of the model, however, by further increasing to 3276 samples with 20 images (Figure C in Appendix) and 6553 samples with 10 images (Figure D in

Appendix) the accuracy of the model increased as the cone of influence widened and reduced its effect.

Through tuning the Batch size from an initial 49 to 4 no significant increase in performance was noted. Most training processes used an epoch size of 5. However, to extract the most accuracy from the final model (Figure D in Appendix) the epoch range was increased to 20, which resulted in an increased accuracy by 2% resulting in a final Validation Accuracy of 77.2%. The Training Process, Confusion Matrices and Cones of Influence of each sample size can be found in Figures A,B,C and D in the Appendix.

Conclusion

Pan Tompkins algorithm for QRS detection is highly effective, the performance review for the sensitivity of the data and the positive predictive value maintains a remarkably high average among the entire patient set defined by the tolerance. The tolerance selected to run in the algorithm sits at 20 samples, when it is set lower, the sensitivity and positive predictive value start to take change majorly. When tolerance is set to 5, the sensitivity lowers to values that would not be accepted for QRS detection. This shows that the false negatives increase when tolerance increases, and the algorithm misses a lot of R-peaks due to this tolerance margin being constricted. There needs to be a golden spot when for accepting what tolerance to use without letting in erroneous looking data sets, that is down to the discretion of the user, as this algorithm is still used universally to this day, there are many ways to improve the algorithm.

For analysis of the Discrete wavelet transform for the detection of QRS complexes it is evident that the performance both in Sensitivity and in Positive Predictive value rest to two major factors, the tolerance of accepting a QRS detection and the mother wavelet used. Firstly, it can be observed form Tables A, B and C in the appendix that as the tolerance of acceptance of QRS detection is increased the calculation for Sensitivity becomes unrealistically high whilst results for PPV suffer. Alternatively, when the tolerance of the QRS detection is decreased the PPV results become unrealistically high with the Sensitivity results suffering. From this it can be concluded that increasing tolerance reduces the false negative values whilst decreasing tolerance results in a decreased false positive rate.

Secondly the selection of the correct mother wavelet is key to detect QRS complexes accurately, as patients within the Arrythmia database have several varying signals it is extremely unlikely that a selected wavelet will accurately detects all QRS complexes for all patients. It is for this reason that the Deep Learning method was explored for classifying different types of ECG signal. If the Validation accuracy could be increased the model could be used to select one of three mother wavelets best suited to detecting that patient’s condition. This would be an exciting prospect for future research into the area of digital signal processing techniques for the detection and classification of ECG signals.

Reference List

[1] 'PhysioBank Annotations', *Physionet.org*. [Online]. Available: https://archive.physionet.org/physiobank/annotations.shtml.

[2] J. Rahul and M. Sora, "Algorithm for QRS complex detection using discrete wavelet transform Biomedical Signal Processing View project." .

[3] M. Jose Da Silva, "Characterization of QRS Complex in ECG Signals Applying Wavelet Transform," Nov. 2015, pp. 86–89, doi: 10.1109/ICMEAE.2015.17.

[4] 'R wave detection in the ECG - MATLAB & Simulink - MathWorks United Kingdom', *Mathworks.com*. [Online].

[5] J. Hill, 'ABC of clinical electrocardiography: Exercise tolerance testing', *BMJ*, vol. 324, no. 7345, pp. 1084–1087, 2002.

[6] K. Schneider and M. Farge, 'Wavelets: Mathematical theory', in *Encyclopedia of Mathematical Physics*, Elsevier, 2006, pp. 426–438.

[7] J. Pan and W. J. Tompkins, 'A real-time QRS detection algorithm', IEEE Trans. Biomed. Eng., vol. 32, no. 3, pp. 230–236, 1985.

Contributions

Algorithm	Name
Pan Tompkins Algorithm	Zeed Alfar
Discrete Wavelet Transform Algorithm	Jack Gibson
Deep Transfer Learning on images generated using Continuous Wavelet Transform	Jack Gibson

Table 2: Contributions for each algorithm

Appendix

Meaning of Beat and Non Beat values

Beat annotations:

Code	Description
N	Normal beat (displayed as "·" by the PhysioBank ATM, LightWAVE, pschart, and psfd)
L	Left bundle branch block beat
R	Right bundle branch block beat
B	Bundle branch block beat (unspecified)
A	Atrial premature beat
a	Aberrated atrial premature beat
J	Nodal (junctional) premature beat
S	Supraventricular premature or ectopic beat (atrial or nodal)
V	Premature ventricular contraction
r	R-on-T premature ventricular contraction
F	Fusion of ventricular and normal beat
e	Atrial escape beat
j	Nodal (junctional) escape beat
n	Supraventricular escape beat (atrial or nodal)
E	Ventricular escape beat
/	Paced beat
f	Fusion of paced and normal beat
Q	Unclassifiable beat
?	Beat not classified during learning

[1]

Non-beat annotations:

Code	**Description**
[	Start of ventricular flutter/fibrillation
!	Ventricular flutter wave
]	End of ventricular flutter/fibrillation
x	Non-conducted P-wave (blocked APC)
(	Waveform onset
)	Waveform end
p	Peak of P-wave
t	Peak of T-wave
u	Peak of U-wave
`	PQ junction
'	J-point
^	(Non-captured) pacemaker artifact
|	Isolated QRS-like artifact
~	Change in signal quality
+	Rhythm change
s	ST segment change
T	T-wave change
*	Systole
D	Diastole
=	Measurement annotation
"	Comment annotation
@	Link to external data

[1]

Pan Tompkins Algorithm Results for Sensitivity and PPV under Differing Tolerance values

Tolerance for 20 Samples

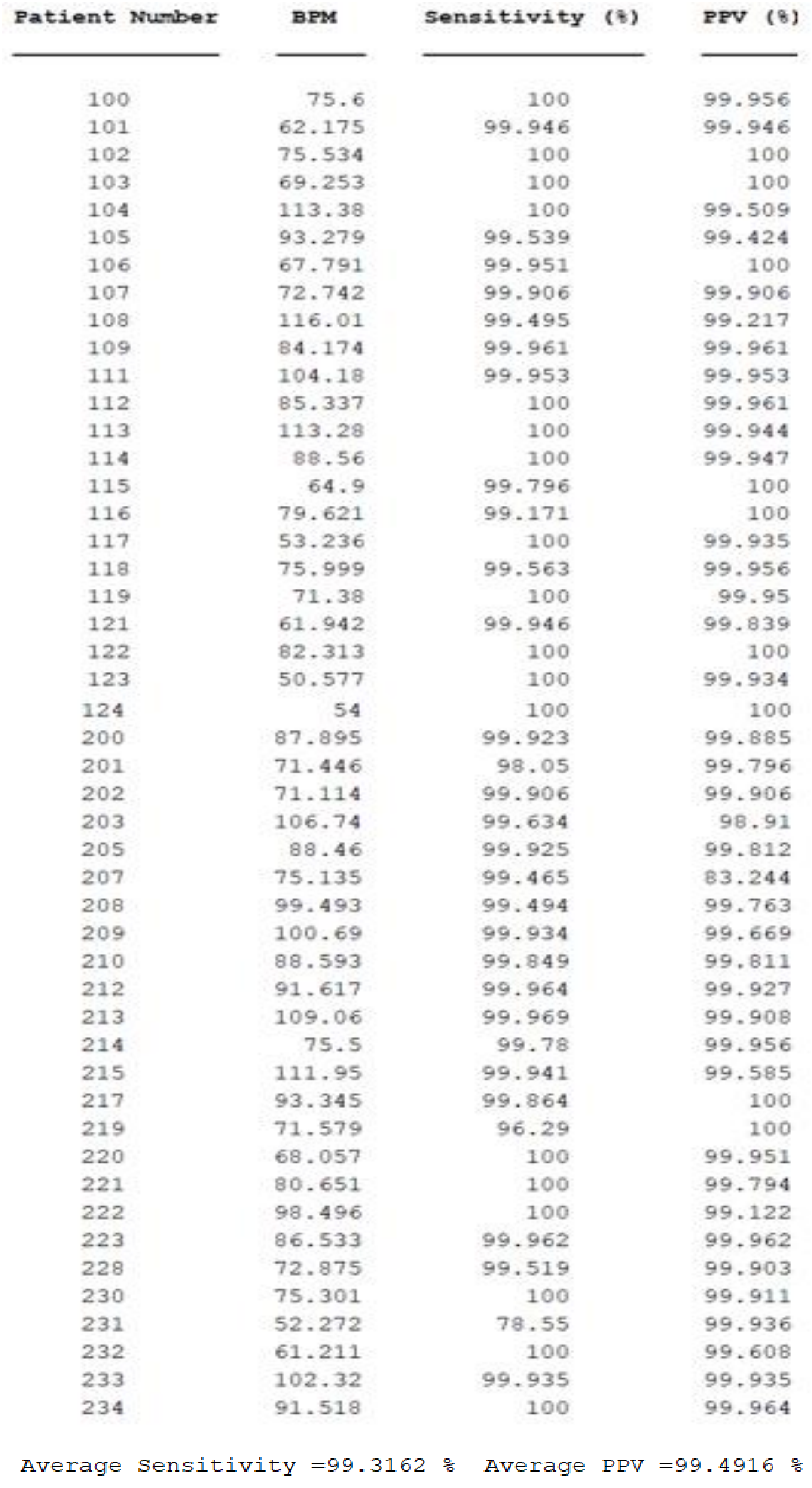
Patient	Sensitivity	PPV
100	99.9560	99.9560
101	99.9463	99.9463
102	99.7653	99.9529
103	99.9520	100
104	96.1164	99.9326
105	99.6104	99.9609
106	99.4079	100
107	99.7660	99.9531
108	99.7730	99.9431
109	99.8025	99.9604
111	99.9529	100
112	100	99.9606
113	99.9442	100
114	97.8179	100
115	100	100
116	98.9635	100
117	100	100
118	100	99.9561
119	98.2888	100
121	99.9463	100
122	100	99.9596
123	100	100
124	99.3205	100
200	99.8462	100
201	96.6378	100
202	99.4382	100
203	99.0927	99.9661
205	99.8870	99.9623
207	98.6021	99.9454
208	95.2929	99.9645
209	100	99.9667
210	99.0566	99.9619
212	100	100
213	99.6001	99.9691
214	99.6460	99.9556
215	99.9405	100
217	99.7433	99.9485
219	100	100
220	100	100
221	99.8352	100
222	98.6709	99.9592
223	99.5777	100
228	99.7564	99.9511
230	100	100
231	100	100
232	100	99.9438
233	99.7726	99.9674
234	99.8910	100

Tolerance for 5 Samples

Patient	Sensitivity	PPV
100	99.9560	100
101	99.9463	99.9463
102	45.3778	99.8965
103	99.9520	100
104	49.3204	99.8686
105	97.0393	99.9598
106	97.7799	100
107	31.9139	100
108	70.3914	99.9194
109	99.0916	100
111	17.8437	100
112	98.6215	100
113	99.9442	100
114	54.2310	100
115	100	100
116	98.2587	100
117	43.9740	100
118	95.0395	100
119	86.3613	100
121	99.8926	100
122	100	100
123	99.9340	100
124	98.2705	100
200	93.1949	100
201	91.1360	100
202	98.8295	100
203	91.4986	99.9633
205	98.8328	100
207	94.7311	99.9432
208	65.2218	99.9481
209	99.2679	99.9665
210	97.5471	100
212	98.9447	100
213	96.4011	100
214	98.9380	99.9553
215	99.3755	100
217	76.1807	99.9326
219	99.5357	100
220	100	100
221	92.9130	100
222	76.3592	100
223	98.7332	100
228	95.3239	99.9489
230	99.6010	100
231	100	100
232	99.4381	100
233	83.3387	99.9610
234	99.8910	100

Discrete Wavelet Transform Results for Sensitivity and PPV under Differing Tolerance values

Table A: Wavelet Performance Results for Tolerance of 54 samples (150ms)



Patient Number	BPM	Sensitivity (%)	PPV (%)
100	75.6	100	99.956
101	62.175	99.946	99.946
102	75.534	100	100
103	69.253	100	100
104	113.38	100	99.509
105	93.279	99.539	99.424
106	67.791	99.951	100
107	72.742	99.906	99.906
108	116.01	99.495	99.217
109	84.174	99.961	99.961
111	104.18	99.953	99.953
112	85.337	100	99.961
113	113.28	100	99.944
114	88.56	100	99.947
115	64.9	99.796	100
116	79.621	99.171	100
117	53.236	100	99.935
118	75.999	99.563	99.956
119	71.38	100	99.95
121	61.942	99.946	99.839
122	82.313	100	100
123	50.577	100	99.934
124	54	100	100
200	87.895	99.923	99.885
201	71.446	98.05	99.796
202	71.114	99.906	99.906
203	106.74	99.634	98.91
205	88.46	99.925	99.812
207	75.135	99.465	83.244
208	99.493	99.494	99.763
209	100.69	99.934	99.669
210	88.593	99.849	99.811
212	91.617	99.964	99.927
213	109.06	99.969	99.908
214	75.5	99.78	99.956
215	111.95	99.941	99.585
217	93.345	99.864	100
219	71.579	96.29	100
220	68.057	100	99.951
221	80.651	100	99.794
222	98.496	100	99.122
223	86.533	99.962	99.962
228	72.875	99.519	99.903
230	75.301	100	99.911
231	52.272	78.55	99.936
232	61.211	100	99.608
233	102.32	99.935	99.935
234	91.518	100	99.964

```
Average Sensitivity =99.3162 %   Average PPV =99.4916 %
```

Table B: Wavelet Performance for Tolerance of 500 Samples (1.4 secs)

Patient Number	BPM	Sensitivity (%)	PPV (%)
100	75.6	100	99.956
101	62.175	100	99.733
102	75.534	100	99.772
103	69.253	100	99.665
104	113.38	100	96.452
105	93.279	100	96.693
106	67.791	100	96.616
107	72.742	100	99.86
108	116.01	100	97.697
109	84.174	100	99.882
111	104.18	100	99.578
112	85.337	100	99.569
113	113.28	100	99.944
114	88.56	100	99.471
115	64.9	100	99.847
116	79.621	99.461	99.626
117	53.236	100	99.74
118	75.999	100	99.435
119	71.38	100	94.89
121	61.942	100	99.307
122	82.313	100	99.96
123	50.577	100	99.934
124	54	100	99.082
200	87.895	100	93.159
201	71.446	100	98.087
202	71.114	100	99.627
203	106.74	100	96.718
205	88.46	100	99.439
207	75.135	99.946	78.314
208	99.493	99.865	97.464
209	100.69	100	98.689
210	88.593	100	98.734
212	91.617	100	99.493
213	109.06	100	98.695
214	75.5	100	98.737
215	111.95	100	98.971
217	93.345	100	96.886
219	71.579	99.956	99.091
220	68.057	100	99.033
221	80.651	100	98.578
222	98.496	100	94.267
223	86.533	100	98.562
228	72.875	100	97.151
230	75.301	100	91.525
231	52.272	100	99.453
232	61.211	100	98.018
233	102.32	100	97.747
234	91.518	100	99.602

Average Sensitivity =99.9839 %   Average PPV =97.9739 %

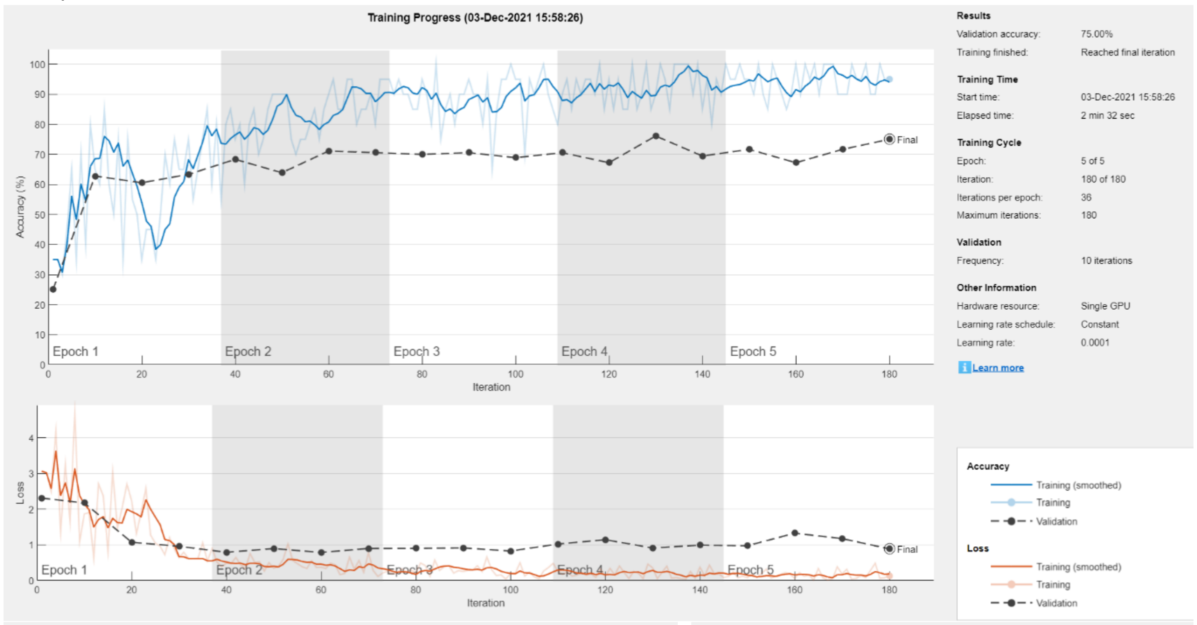
Table C: Wavelet Performance for Tolerance of 5 Samples (14ms)

Patient Number	BPM	Sensitivity (%)	PPV (%)
100	75.6	100	100
101	62.175	99.786	100
102	75.534	82.625	100
103	69.253	100	100
104	113.38	98.385	100
105	93.279	98.924	100
106	67.791	99.26	100
107	72.742	99.579	100
108	116.01	82.492	99.864
109	84.174	98.776	100
111	104.18	99.058	100
112	85.337	100	100
113	113.28	100	100
114	88.56	46.33	100
115	64.9	99.694	100
116	79.621	99.129	100
117	53.236	34.267	100
118	75.999	99.476	100
119	71.38	78.661	100
121	61.942	99.946	100
122	82.313	99.96	100
123	50.577	100	100
124	54	99.321	100
200	87.895	99.154	100
201	71.446	96.7	99.948
202	71.114	99.719	100
203	106.74	95.975	100
205	88.46	99.285	100
207	75.135	98.233	92.958
208	99.493	70.132	100
209	100.69	99.867	100
210	88.593	99.283	100
212	91.617	99.964	100
213	109.06	96.247	100
214	75.5	99.162	100
215	111.95	99.851	100
217	93.345	66.274	100
219	71.579	92.1	100
220	68.057	99.951	100
221	80.651	100	100
222	98.496	99.839	99.96
223	86.533	99.232	100
228	72.875	97.885	100
230	75.301	99.956	100
231	52.272	78.55	100
232	61.211	99.944	100
233	102.32	85.492	100
234	91.518	100	100

Average Sensitivity =93.5096 %   Average PPV =99.8486 %

Deep Learning Results

Figure A: Training Process, Spectrogram with Cone of Influence and Confusion Matrix **for 500 Samples** per image- 10 images per ECG signal – 5 Epoch



Results

Validation accuracy:	75.00%
Training finished:	Reached final iteration

Training Time

Start time:	03-Dec-2021 15:58:26
Elapsed time:	2 min 32 sec

Training Cycle

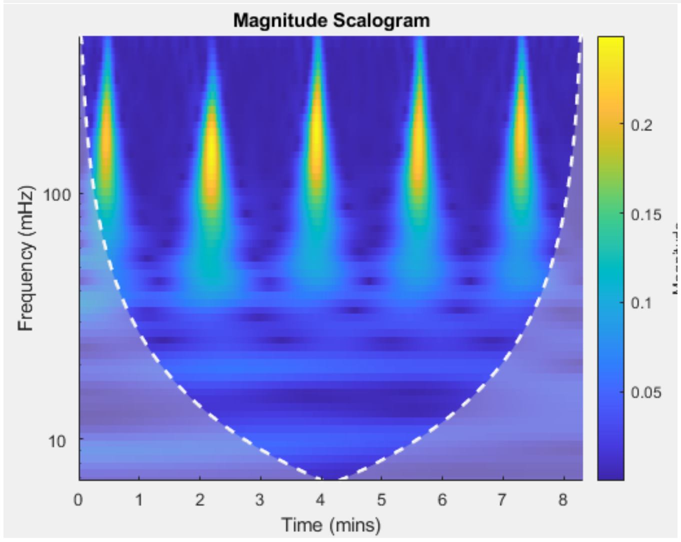
Epoch:	5 of 5
Iteration:	180 of 180
Iterations per epoch:	36
Maximum iterations:	180

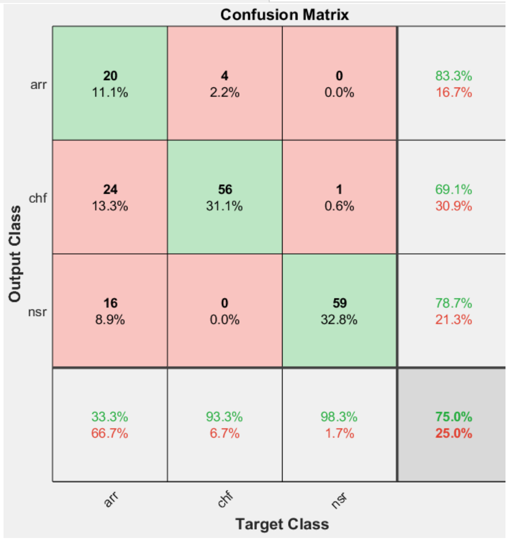
Validation

Frequency:	10 iterations

Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.0001



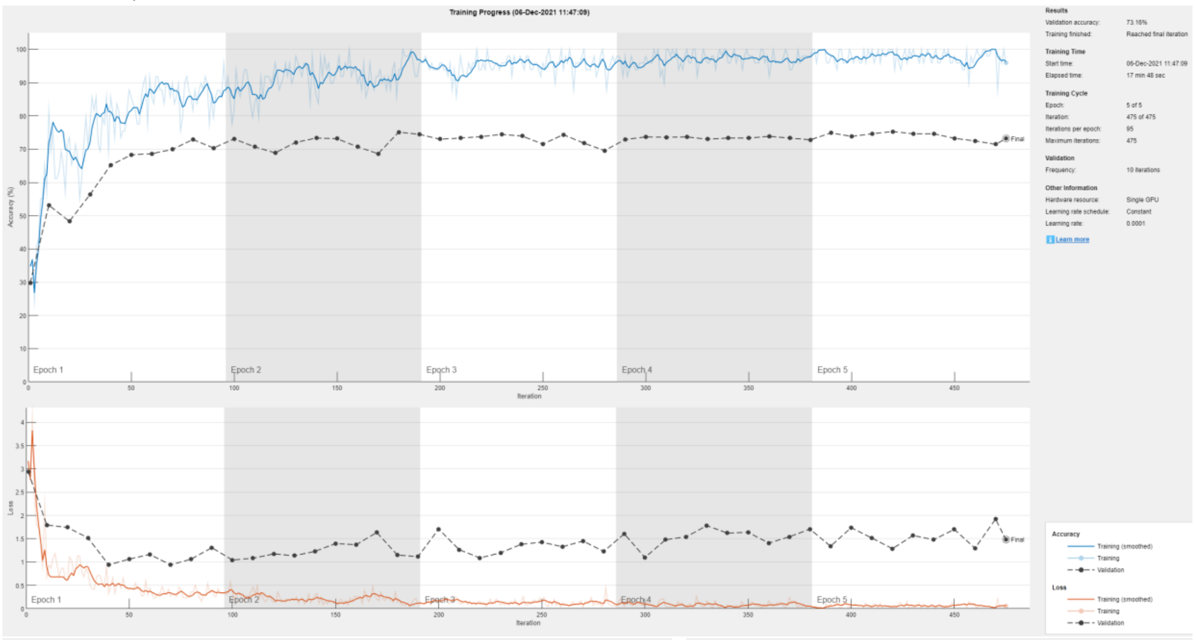


Confusion Matrix

Output Class \ Target Class	arr	chf	nsr	
arr	20 11.1%	4 2.2%	0 0.0%	83.3% 16.7%
chf	24 13.3%	56 31.1%	1 0.6%	69.1% 30.9%
nsr	16 8.9%	0 0.0%	59 32.8%	78.7% 21.3%
	33.3% 66.7%	93.3% 6.7%	98.3% 1.7%	**75.0%** **25.0%**

Validation Accuracy 75%

Figure B: Training Process, Spectrogram with Cone of Influence and Confusion Matrix for 1000 Samples per image- 65 images per ECG signal – 5 Epoch



Results

Validation accuracy:	73.16%
Training finished:	Reached final iteration

Training Time

Start time:	06-Dec-2021 11:47:09
Elapsed time:	17 min 48 sec

Training Cycle

Epoch:	5 of 5
Iteration:	475 of 475
Iterations per epoch:	95
Maximum iterations:	475

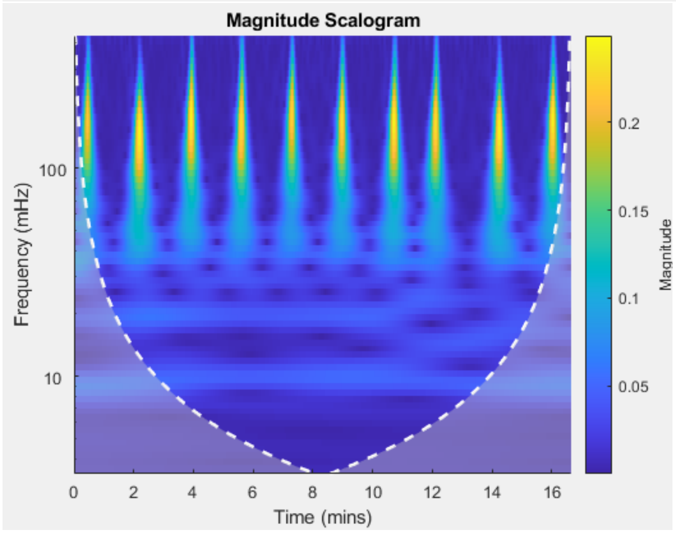
Validation

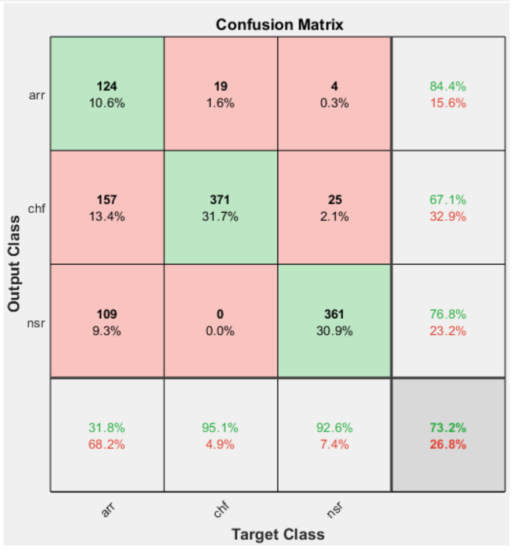
Frequency:	10 iterations

Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Learn more



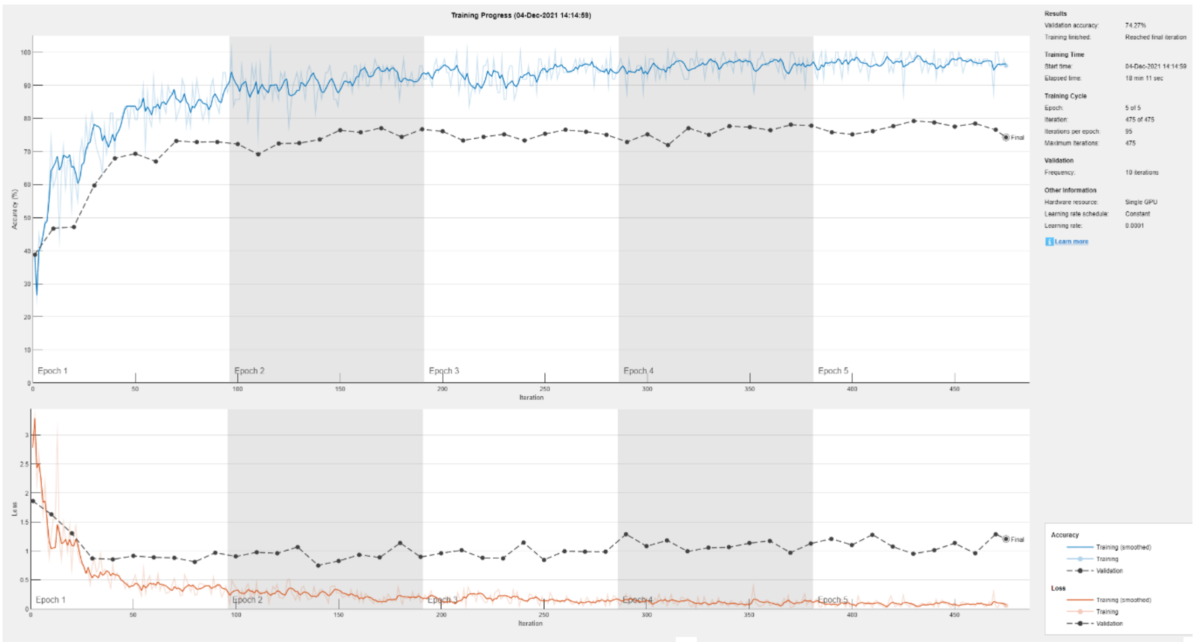


Output Class \ Target Class	arr	chf	nsr	
arr	124 10.6%	19 1.6%	4 0.3%	84.4% 15.6%
chf	157 13.4%	371 31.7%	25 2.1%	67.1% 32.9%
nsr	109 9.3%	0 0.0%	361 30.9%	76.8% 23.2%
	31.8% 68.2%	95.1% 4.9%	92.6% 7.4%	73.2% 26.8%

Do again for this

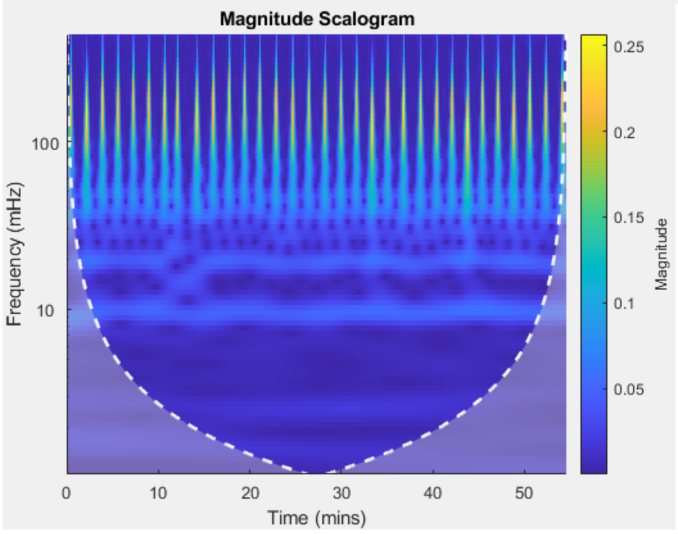
Validation Accuracy 73.16%

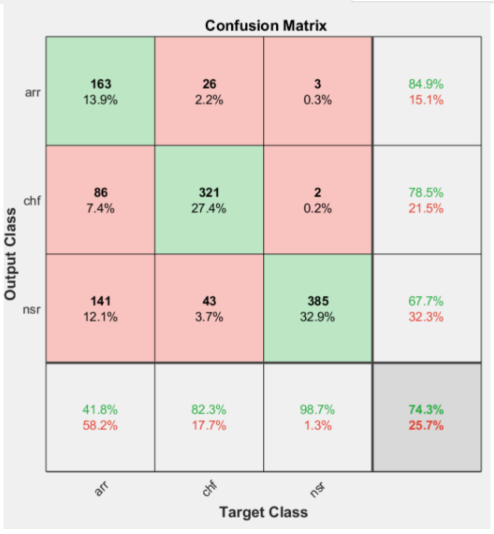
<u>Figure C: Training Process, Spectrogram with Cone of Influence and Confusion Matrix **for 3276 Samples** per image- 20 images per ECG signal – 5 Epoch</u>



Results	
Validation accuracy:	74.27%
Training finished:	Reached final iteration
Training Time	
Start time:	04-Dec-2021 14:14:59
Elapsed time:	18 min 11 sec
Training Cycle	
Epoch:	5 of 5
Iteration:	475 of 475
Iterations per epoch:	95
Maximum iterations:	475
Validation	
Frequency:	10 iterations
Other Information	
Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Learn more



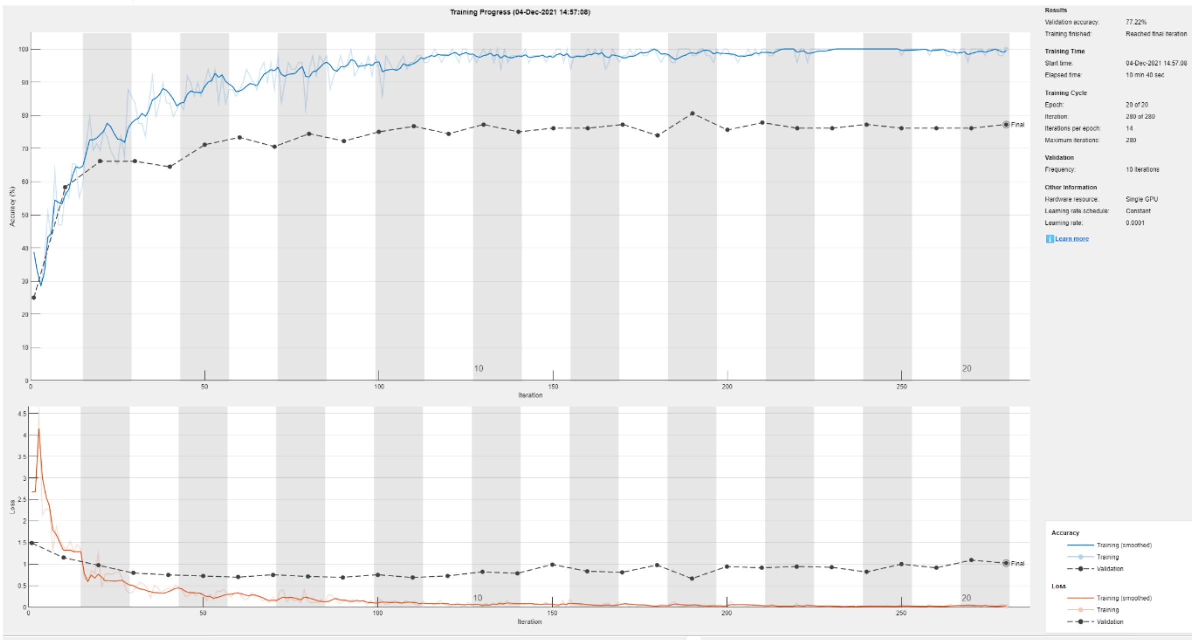


Confusion Matrix

Output Class \ Target Class	arr	chf	nsr	
arr	163 13.9%	26 2.2%	3 0.3%	84.9% 15.1%
chf	86 7.4%	321 27.4%	2 0.2%	78.5% 21.5%
nsr	141 12.1%	43 3.7%	385 32.9%	67.7% 32.3%
	41.8% 58.2%	82.3% 17.7%	98.7% 1.3%	74.3% 25.7%

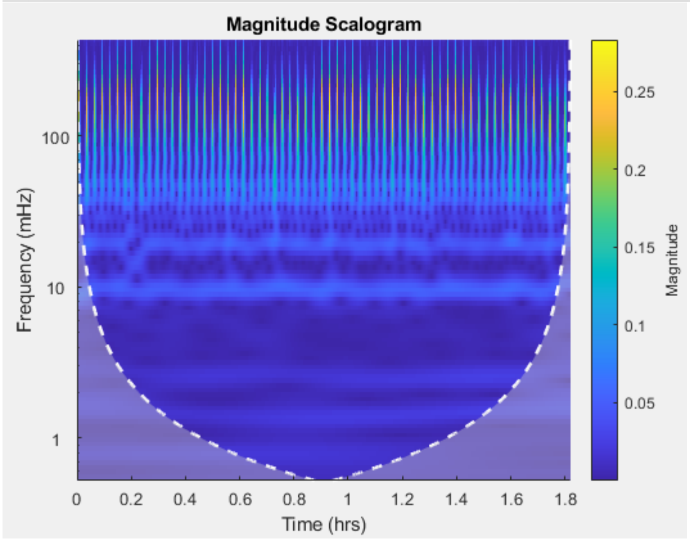
Validation Accuracy 74.27%

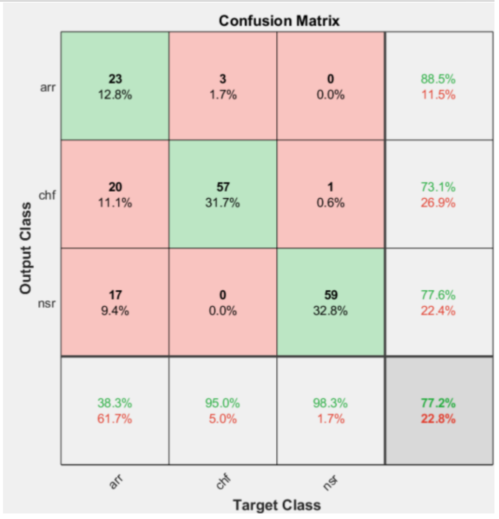
Figure D: Training Process, Spectrogram with Cone of Influence and Confusion Matrix **for 6553 Samples** per image- 10 images per ECG signal – 20 Epoch



Results	
Validation accuracy:	77.22%
Training finished:	Reached final iteration
Training Time	
Start time:	04-Dec-2021 14:57:08
Elapsed time:	10 min 40 sec
Training Cycle	
Epoch:	20 of 20
Iteration:	280 of 280
Iterations per epoch:	14
Maximum iterations:	280
Validation	
Frequency:	10 iterations
Other Information	
Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Learn more





Confusion Matrix

Output Class \ Target Class	arr	chf	nsr	
arr	23 12.8%	3 1.7%	0 0.0%	88.5% 11.5%
chf	20 11.1%	57 31.7%	1 0.6%	73.1% 26.9%
nsr	17 9.4%	0 0.0%	59 32.8%	77.6% 22.4%
	38.3% 61.7%	95.0% 5.0%	98.3% 1.7%	77.2% 22.8%

Validation Accuracy 77.22%

Code For Loading Dataset

Dr Pardis Biglarbeigi

```
% add the data URl to the code:

% will download the zip file of the package
wfdb_url='https://physionet.org/physiotools/matlab/wfdb-app-matlab/wfdb-
app-toolbox-0-10-0.zip';
% will save the web URL in the directory (you can change the path to it)
outfilename = websave('wfdb-app-toolbox-0-10-0.zip',wfdb_url);
% will unzip the zip file
unzip('wfdb-app-toolbox-0-10-0.zip');
% will add the path of teh unzipped folder to the code
addpath ('mcode')

% download the Records file and find the names in the file:
urlwrite('https://physionet.org/files/mitdb/1.0.0/RECORDS', 'Records.txt');
Records = load ('Records.txt', '-ascii'); % these would be the name of all
the files
% this makes a new folder in the directory to save the data in
mkdir('data')
for i = 1:length(Records)
    %each file name
    file = num2str(Records(i));
    %load the data
     [signal,~,~]=rdsamp(['mitdb/' file]);
     signal = signal(: , 1); % save one channel of the signal
     %load the annotations
       [ann , anntype] = rdann(['mitdb/' file], 'atr');
       %create a structured dataset for each patient
       eval(['pat_', file, '.sig = signal' ]) ;
     eval(['pat_', file, '.ann = ann' ]) ;
     eval(['pat_', file, '.annType = anntype' ]) ;
     % save the data in data folder made in line 16
     save(['data/pat_',  file, '.mat'], ['pat_', file])

end
```