SS_ECG_IA

August 16, 2022

1 Neural network for ECG analysis

Javier Serrano Molina and Erin Christy McKiernan Facultad de ciencias, UNAM

1.1 Setting up the notebook

We begin by setting up the Jupyter notebook and importing the Python modules needed for plotting figures, create animations, etc. We include commands to view plots in the Jupyter notebook, and to create figures with good resolution and large labels. These commands can be customized to produce figures with other specifications.

[1]:
```
# Imports python libraries
import numpy as np
import random as rd
import wave
import sys
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from scipy.signal import butter, lfilter, filtfilt #for filtering data
from statistics import stdev
sys.path.insert(1, r'./../functions') # add to pythonpath

# commands to create high-resolution figures with large labels
%config InlineBackend.figure_formats = {'png', 'retina'}
plt.rcParams['axes.labelsize'] = 16 # fontsize for figure labels
plt.rcParams['axes.titlesize'] = 18 # fontsize for figure titles
plt.rcParams['font.size'] = 14 # fontsize for figure numbers
plt.rcParams['lines.linewidth'] = 1.4 # line width for plotting
```

[2]:
```
#comdands needed to import the data and the metadata.
from IPython.display import display
import os
import shutil
import posixpath
import wfdb
```

1.2 Defining a function to retrieve the data

We define a function called ecg that opens the recordings with a given path. The outcome is a couple of arrays that correspond to the time variable given in seconds and the data given in μV.

[3]:
```
#Function that extracts the number of recording channels, sampling rate, time␣
 ↪and signal
#variable is the path and filename of the .wav file
def ecg(variable):
    record = wave.open(variable, 'r') # load the data

    # Get the number of channels, sample rate, etc.
    numChannels = record.getnchannels() #number of channels
    numFrames = record.getnframes() #number of frames
    sampleRate = record.getframerate() #sampling rate
    sampleWidth = record.getsampwidth()

    # Get wave data
    dstr = record.readframes(numFrames * numChannels)
    waveData = np.frombuffer(dstr, np.int16)

    # Get time window
    timeECG = np.linspace(0, len(waveData)/sampleRate, num=len(waveData))

    return timeECG, waveData
```

1.3 Re-defining the previous functions

We define the functions that were obtained in the other notebooks. If there is any doubt, about how they were obtained, please review the notebook 01-Basics (for detecta_maximos_locales) or SS_ECG(for the rest of functions).

[4]:
```
def detecta_maximos_locales(timeECG, waveData, threshold_ratio=0.7):
    # If not all the R peaks are detected, lower the threshold_ratio
    # If components that are not R peaks (like T waves) are detected, higher␣
 ↪the threshold_ratio

    if len(timeECG) != len(waveData): #Raises an error if the two arrays have␣
 ↪different lengths
        raise Exception("The two arrays have different lengths.")

    interval = max(waveData) - min(waveData)
    threshold = threshold_ratio*interval + min(waveData)
    maxima = []
    maxima_indices = []
    mxs_indices = []
    banner = False
```

```
    for i in range(0, len(waveData)):

        if waveData[i] >= threshold:#If a threshold value is surpassed,
            # the indices and values are saved
            banner = True
            maxima_indices.append(i)
            maxima.append(waveData[i])

        elif banner == True and waveData[i] < threshold: #If the threshold␣
 ↪value is crossed
            # the index of the maximum value in the original array is saved
            index_local_max = maxima.index(max(maxima))
            mxs_indices.append(maxima_indices[index_local_max])
            maxima = []
            maxima_indices = []
            banner = False

    return mxs_indices
```

[5]:
```
#Functions to get the Q and S peaks that correspond to any given R peak
#f: array of voltages
#x0: index of an R peak.
#w: time window in which the corresponding marker will be searched

def last_min(f,x0, w=18):
    #This function searches for a minimum in the time window previous to the R␣
 ↪peak (x0)
    #If the R peak is too close to the begining of the recording, the function␣
 ↪will search in the available window.
    if x0<w:
        x=np.argmin(f[:x0])
        return(x+x0)
    else:
        x=np.argmin(f[x0-w:x0])
        return(x+x0-w)
def next_min(f,x0, w=18):
    #This function searches for a minimum in the time window after the R␣
 ↪peak(x0)
    x=np.argmin(f[x0:x0+w])
    return(x+x0)
```

[6]:
```
#Function that obtains and plots the QRS complexes in a given recording
#timeECG: Time array in seconds
#waveData: Voltage array in micro-volts$
#x2: Filtered voltage array after fourier transform and reconstruction
#threshold_ratio: Relative threshold. If exceded, an R peak will be searched
```

```
def QRS(timeECG,waveData,x2,threshold_ratio,fs, plot=True):
    #Seaches the indexes of R peaks in the filtered signal
    mxs_indices = detecta_maximos_locales(timeECG[2:], x2[2:],threshold_ratio)

    #Empty arrays are declared for the indexes of QRS markers
    Q=[]
    R=[]
    S=[]
    #Array of lenghtds from individual QRS complexes
    qrs_lenght=[]

    #For every R peak, the QRS complex is identified and saved
    for r in mxs_indices:
        r+=2
        Q.append(last_min(waveData,r,w=int(fs*0.08)))
        S.append(next_min(waveData,r,w=int(fs*0.08)))
        qrs_lenght.append((S[-1]-Q[-1])/fs)
        qrs_complex=waveData[Q[-1]:S[-1]]
        if S[-1]>Q[-1]+1:
            R.append(np.argmax(qrs_complex)+Q[-1])
    #An array of results is declared
    r=[np.mean(qrs_lenght),np.max(qrs_lenght),np.min(qrs_lenght),np.
 ↪std(qrs_lenght)]

    # Plotting EMG signal
    if plot:
        plt.figure(figsize=(18,6))
        plt.xlabel(r'time (s)')
        plt.ylabel(r'voltage ($\mu$V)')
        plt.plot(timeECG,waveData, 'b')
        #plt.plot(timeECG,2*x2.real,'g')
        plt.scatter(timeECG[R], waveData[R], color='r')
        plt.scatter(timeECG[Q], waveData[Q], color='g')
        plt.scatter(timeECG[S], waveData[S], color='m')
        #plt.xlim(timeECG[0],timeECG[0]+4)
        plt.title(file.split("/")[1]+"- "+str(int(timeECG[0]//20)))
        plt.show();
    return Q,R,S,r
```

[7]:
```
#The recording is obtained from the file path and filtered using the fast␣
 ↪Fourier transform(fft)
#file: File path
#fm: Frecuency minimum. Components of low frequency are deleted.
#fM: Frecuency maximum. Components of high frecuency are deleted.
#threshold_ratio: Relative threshold that if surpased, an R peak will be␣
 ↪searched.
#r: Array of results. Every result corresponds to a sample.
```

```
#Every sample has 7 values: mean qrs lenght, max qrs lenght, min qrs lenght,␣
 ↪mean heart rate, heart reate std, max heart rate, min heart rate.

def R_fourier(file,fm,fM,threshold_ratio=0.7,plot=True):
    record = wfdb.rdrecord(file, sampto = 15000)
    waveData = record.p_signal[:,0]
    timeECG = np.array([i/record.fs for i in range(0, len(waveData))])

    #The Fourier transform is generated through the fft function
    X=np.fft.fft(waveData)
    N = len(X)
    n = np.arange(N)
    T = N*(timeECG[1]-timeECG[0])
    freq = n/T

    #The Foruier transform is filtered and the function is reconstructed using␣
 ↪the inverse fast fourier transform(ifft)
    xc=X.copy()
    xc[freq<fm]=0
    xc[freq>fM]=0
    x2=np.fft.ifft(xc)

    #Spliting in segments of 20s
    #tt: Total time
    tt=timeECG[-1]
    #spm: Samples per measurment
    spm=int(tt//20)
    r=[]

    #for every sample, the corresponding analysis is made
    for i in range(0,spm):
        spm_i=i*record.fs*20
        spm_f=spm_i+record.fs*20
        Q,R,S,a=QRS(timeECG[spm_i:spm_f],waveData[spm_i:spm_f],x2[spm_i:
 ↪spm_f],threshold_ratio,record.fs,plot=plot)
        hr=[]

        for j in range(1,len(R)):
            heart_rate=60/(timeECG[R[j]]-timeECG[R[j-1]])
            hr.append(heart_rate)
        r.append([a[0],a[1],a[2],np.mean(hr),np.std(hr),np.max(hr),np.min(hr)])

    return(r)
```

1.4 QRS complex: Normal vs Arrythmia

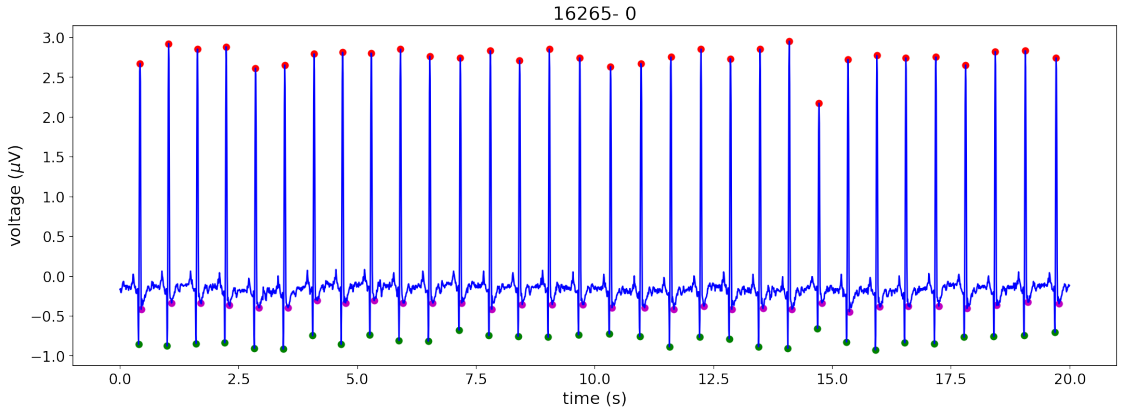
Using the set of data previously imported and analyzed in the past notebooks we will find the QRS complex and analyze it. Based on previous observations we would expect to be able to identify arrythmias from the R-R segment variation and QRS complex duration.

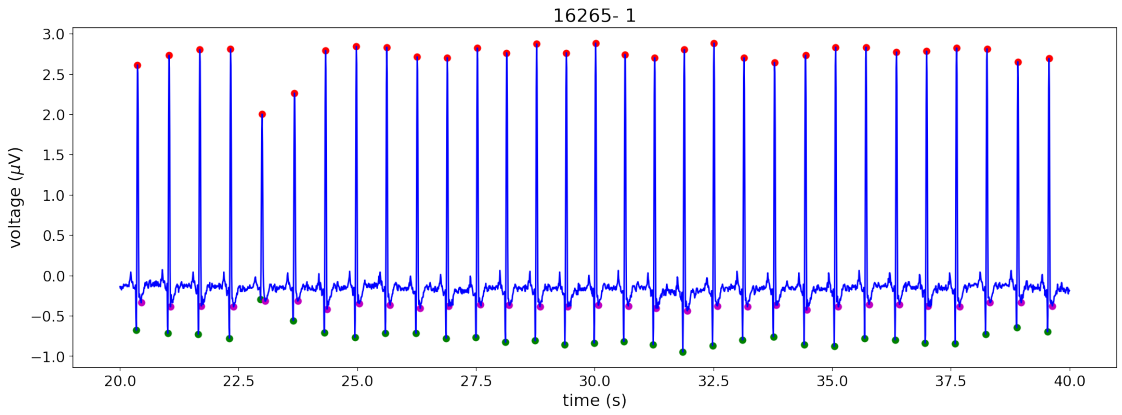
[8]:
```
#All the files corresponding to patients with arrhythmia are imported, graphed␣
 ↪and analyzed
file=open("mit-bih-arrhythmia-database-1.0.0/RECORDS","r")
a=file.read()
b=a.split(a[3])
b.pop()
b=np.array(b)
ra=[[0.1,0.1,0.1,60,6,60,60]]
#Notice that some recordings were erased
for record in␣
 ↪b[[0,1,3,5,6,11,12,14,15,16,19,21,22,24,26,27,30,32,33,35,36,37,38,41,42,43,44,45,47]]:
 ↪
#for record in b:
    file="mit-bih-arrhythmia-database-1.0.0/"+record
    ra=np.append(ra,R_fourier(file,5,35,threshold_ratio=0.7,plot=False),axis=0)
ra=np.delete(ra,[0],axis=0)
ra=np.delete(ra,[28,38,47,48,54],axis=0)
```

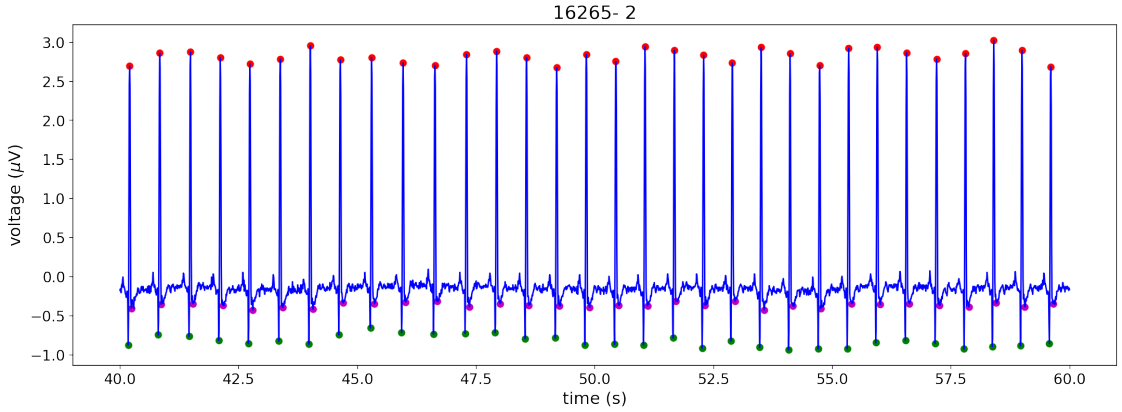
```
/Users/javierserranomolina/miniconda3/envs/IA/lib/python3.7/site-
packages/ipykernel_launcher.py:42: RuntimeWarning: divide by zero encountered in
double_scalars
/Users/javierserranomolina/miniconda3/envs/IA/lib/python3.7/site-
packages/numpy/core/_methods.py:202: RuntimeWarning: invalid value encountered
in subtract
  x = asanyarray(arr - arrmean)
```

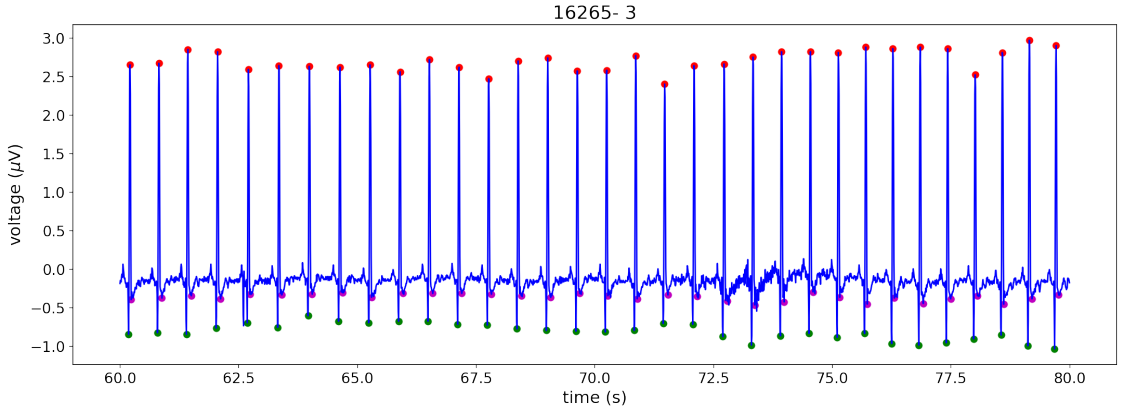
Discrimination criteria	Normal samples	Arrythmia samples
QRS complex not clear	19088-A(13), 19088-B(13), 19090-A(14), 19090-B(14), 19090-E(14), 19093-A(15), 19093-B(15), 19114-A(16), 19114-B(16)	104(4),108(8), 122(20), 200(23), 207-A(28), 208(29), 221(39)
Deformed QRS complex	16483-E(4), 16539-B(5), 16786-C(7), 18184-A(12), 18184-B(12), 18184-C(12), 18184-D(12)	102(2), 107(7), 108(8), 111(10),114(13), 118(17), 119(18), 202(25)
Noise Excess/Artifacts in sample	16272-A(1), 16273-B(2), 16795-A(8), 16795-C(8), 18177-C(11), 19830(17)	104(4), 109(9), 203-A(26), 207(28), 210-A(31), 214(34), 222(40), 223-B(41), 228-A(42), 232-A(45), 233(46)

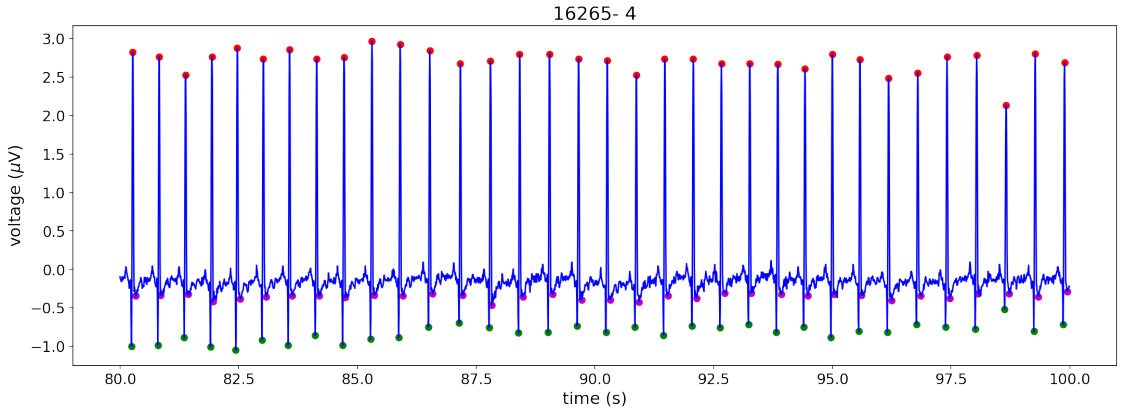
[9]:
```
#All the files corresponding to normal patients are imported, graphed and␣
 ↪analyzed
file=open("mit-bih-normal-sinus-rhythm-database-1.0.0/RECORDS","r")
a=file.read()
b=a.split(a[5])
b.pop()
b=np.array(b)
#Notice that some recordings were deleted
rn=[[0.1,0.1,0.1,60,6,60,60]]
for record in b[0:17]:
    file="mit-bih-normal-sinus-rhythm-database-1.0.0/"+record
    rn=np.append(rn,R_fourier(file,5,35,threshold_ratio=0.6, ),axis=0)
rn=np.delete(rn,[0],axis=0)
rn=np.
 ↪delete(rn,[5,11,24,26,37,40,42,57,60,61,62,63,65,66,70,71,74,75,76,80,81],axis=0)
```

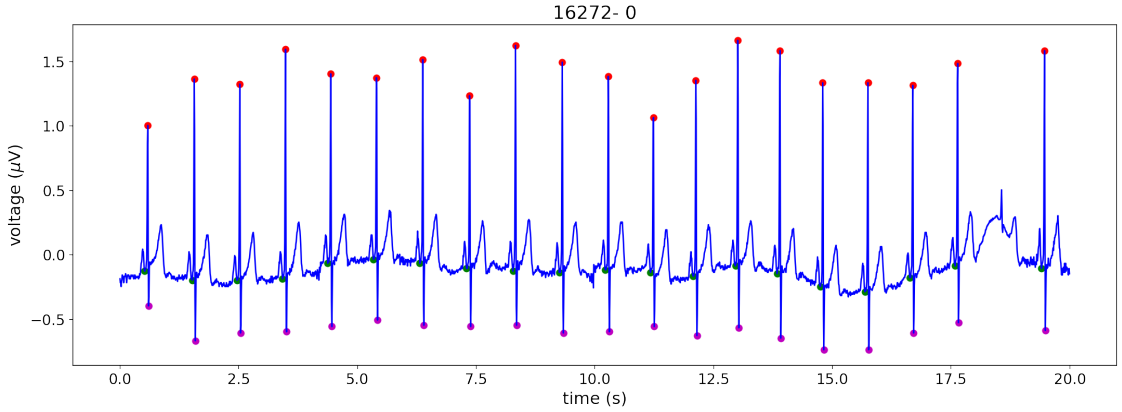


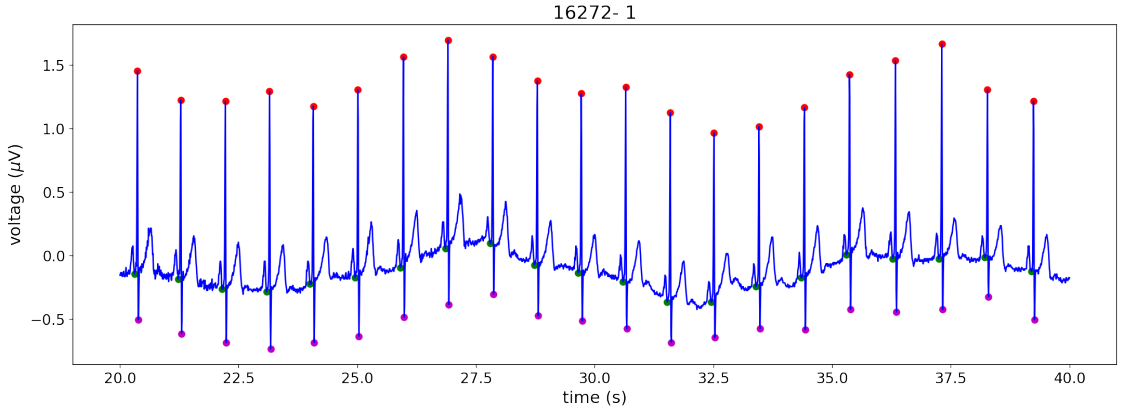


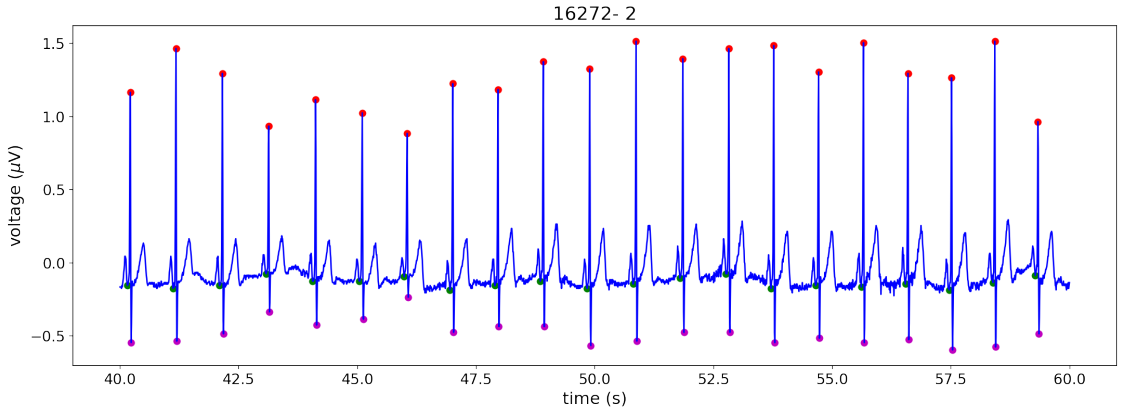

16265- 2
voltage (μV)
time (s)

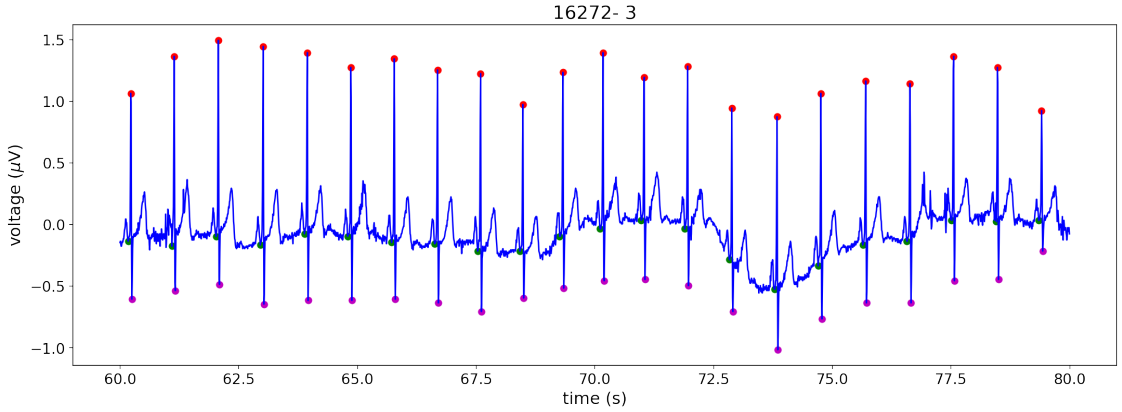

16265- 3
voltage (μV)
time (s)

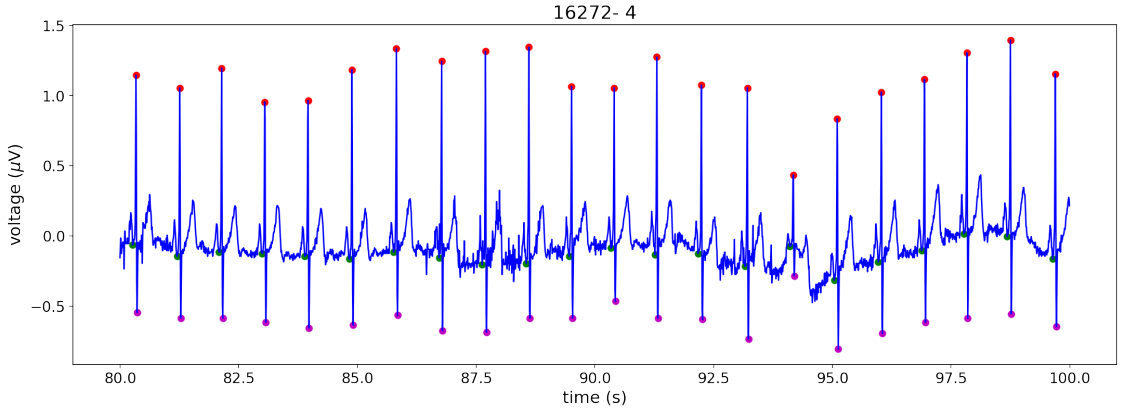

16265- 4
voltage (μV)
time (s)

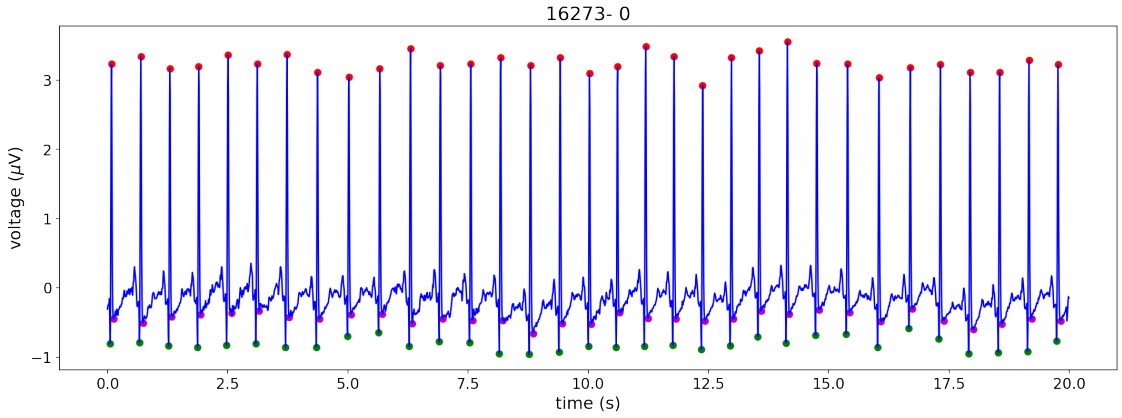

16272- 0
voltage (μV)
time (s)

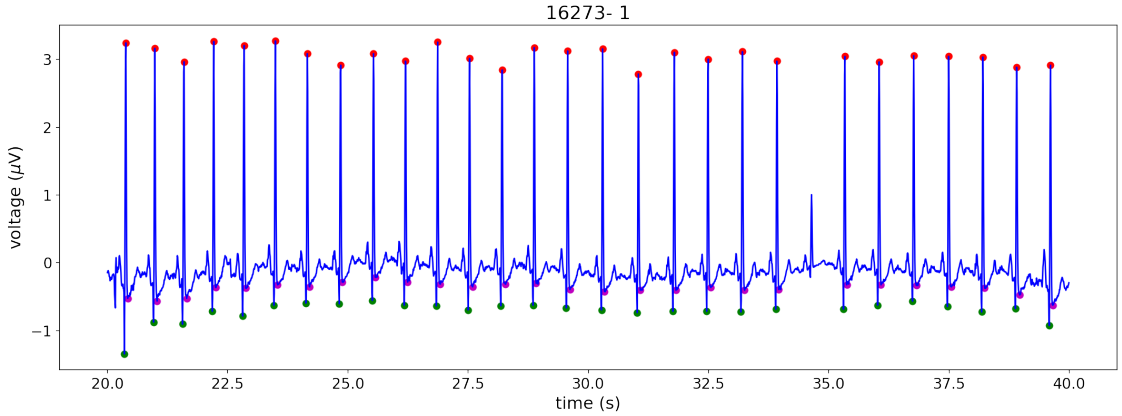

16272- 1
voltage (μV)
time (s)

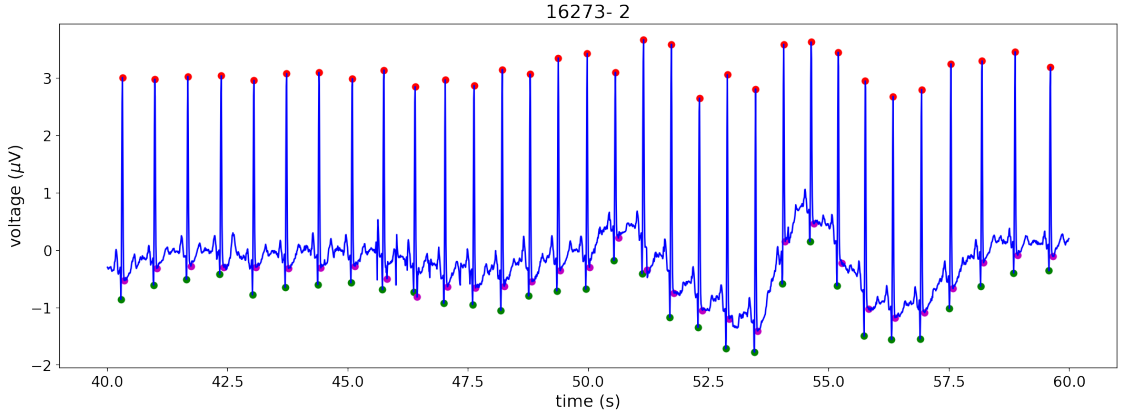

16272- 2
voltage (μV)
time (s)

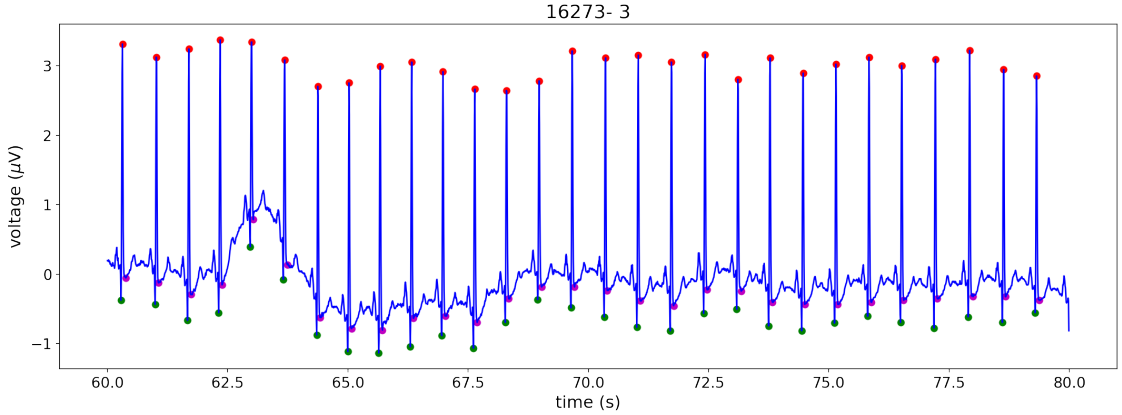

16272- 3
voltage (μV)
time (s)

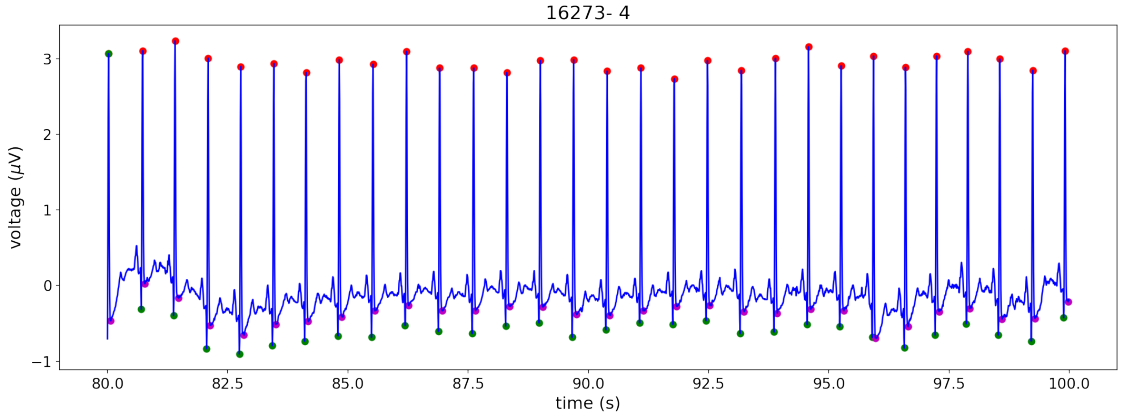

16272- 4
voltage (μV)
time (s)

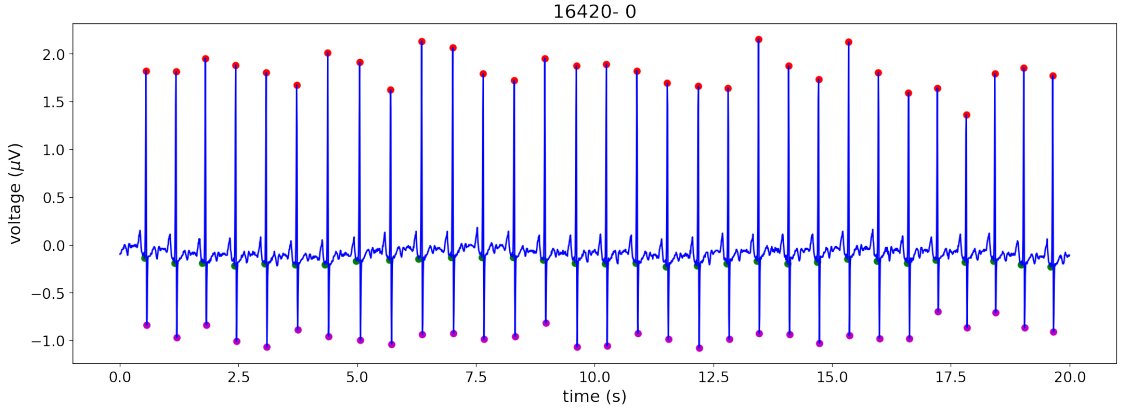

16273- 0
voltage (μV)
time (s)

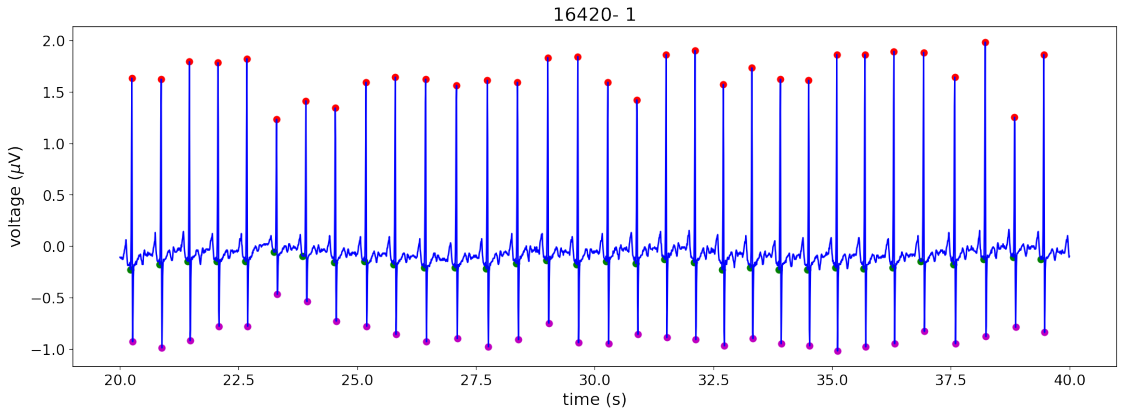

16273- 1
voltage (μV)
time (s)

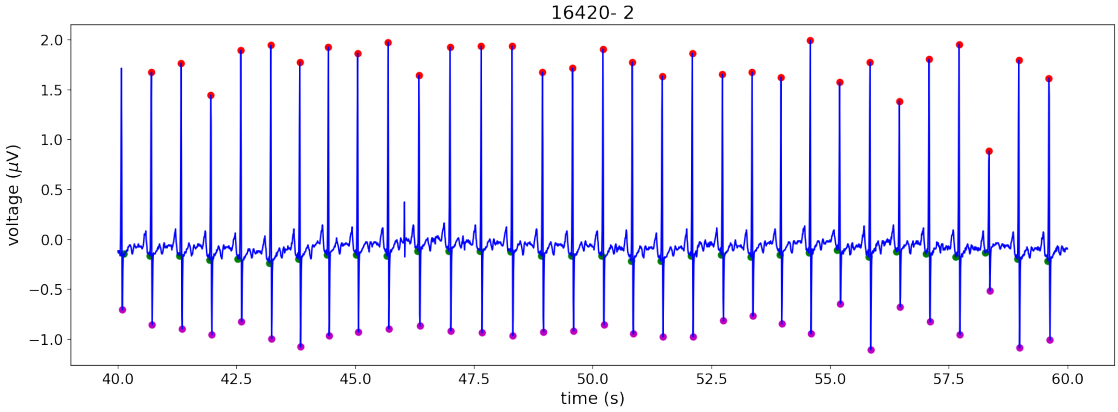

16273- 2
voltage (μV)
time (s)

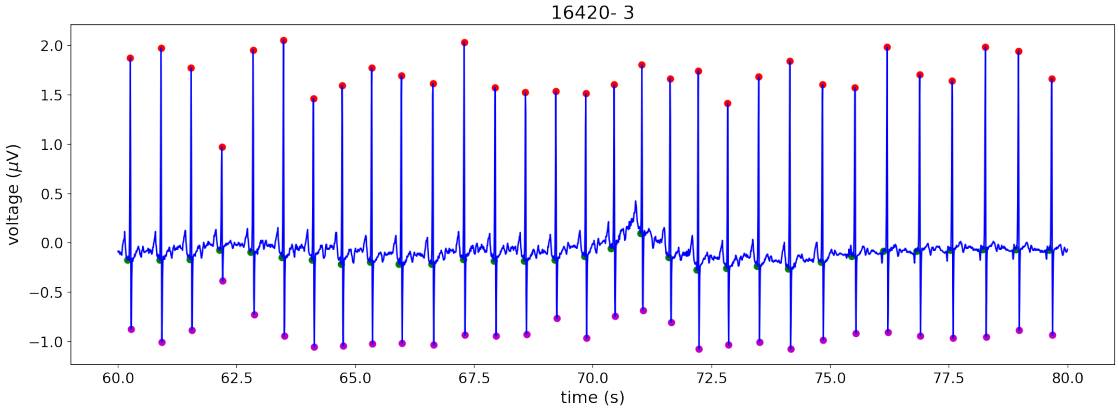

16273- 3
voltage (μV)
time (s)

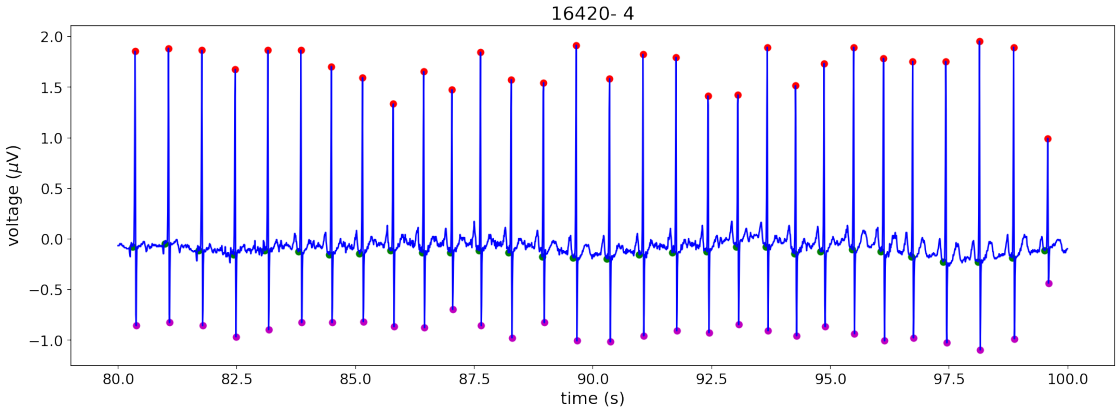

16273- 4
voltage (μV)
time (s)

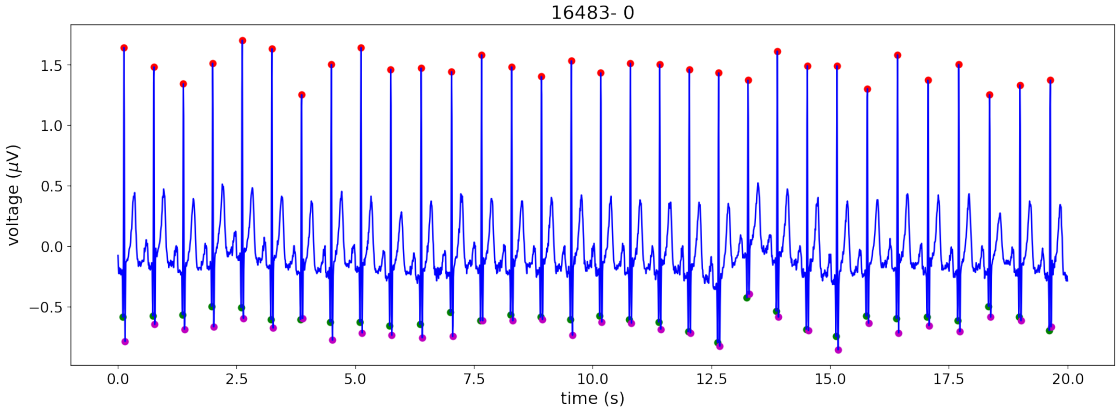

16420- 0
voltage (μV)
time (s)

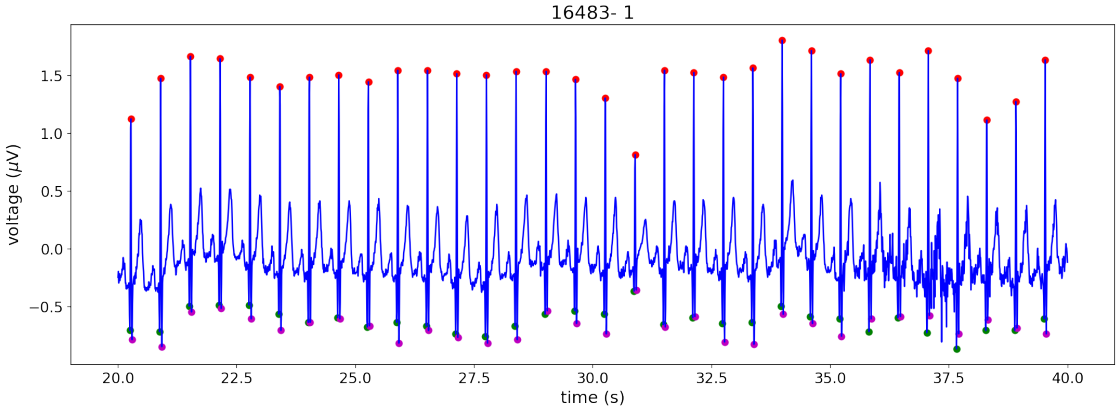

16420- 1
voltage (μV)
time (s)

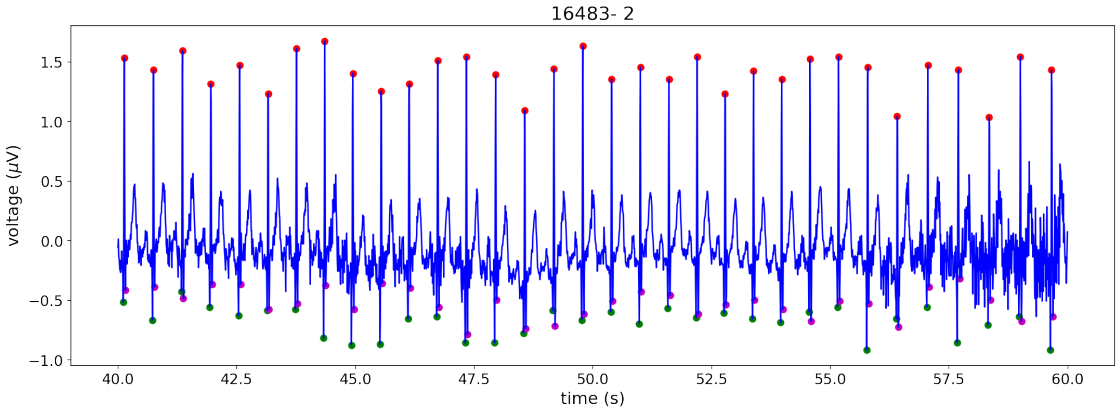

16420- 2
voltage (μV)
time (s)

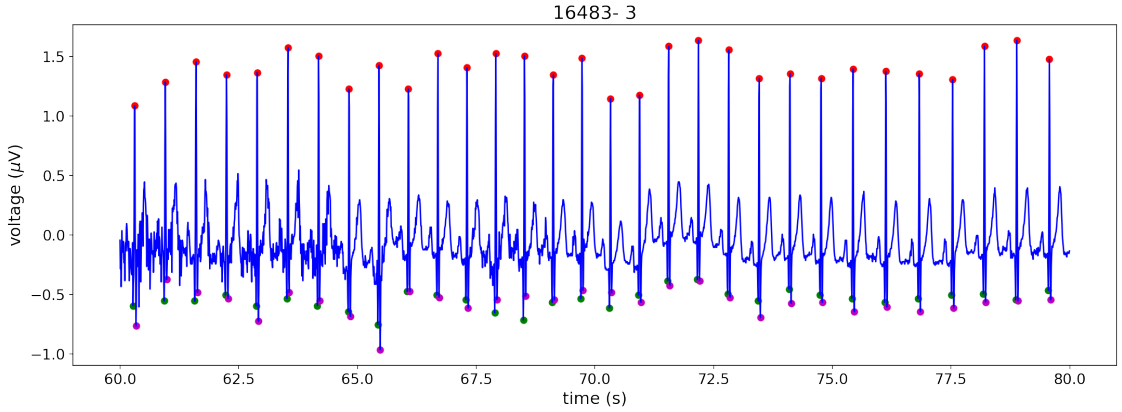

16420- 3
voltage (μV)
time (s)

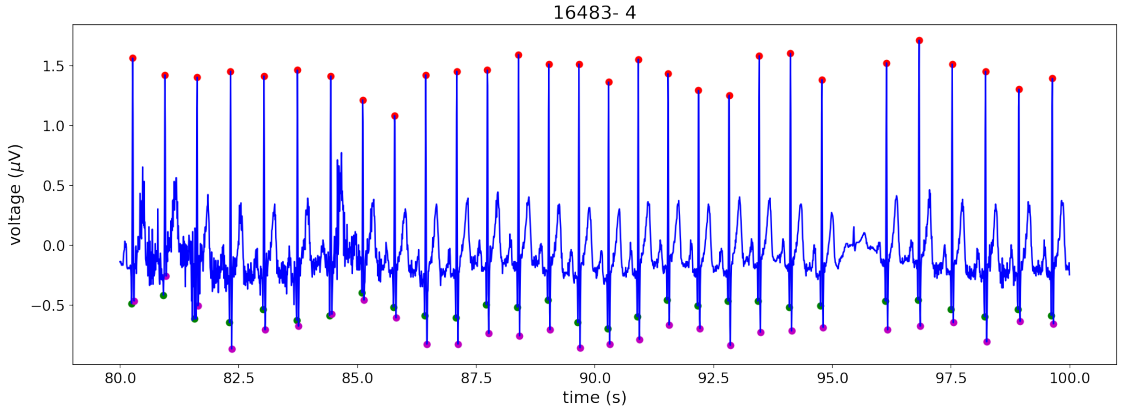

16420- 4
voltage (μV)
time (s)

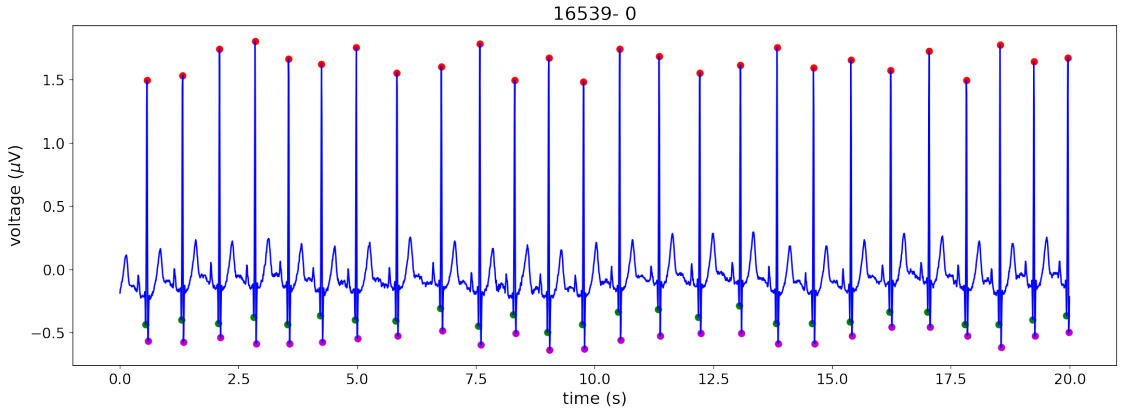

16483- 0
voltage (μV)
time (s)

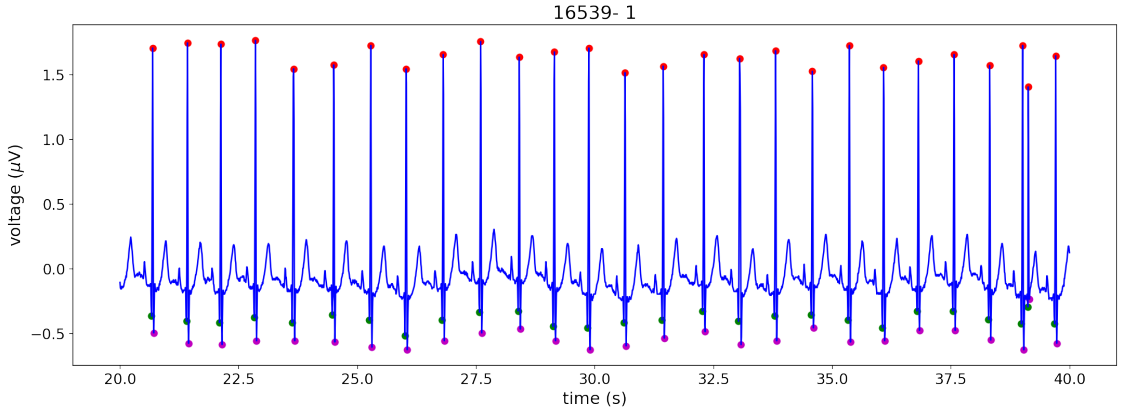

16483- 1
voltage (μV)
time (s)

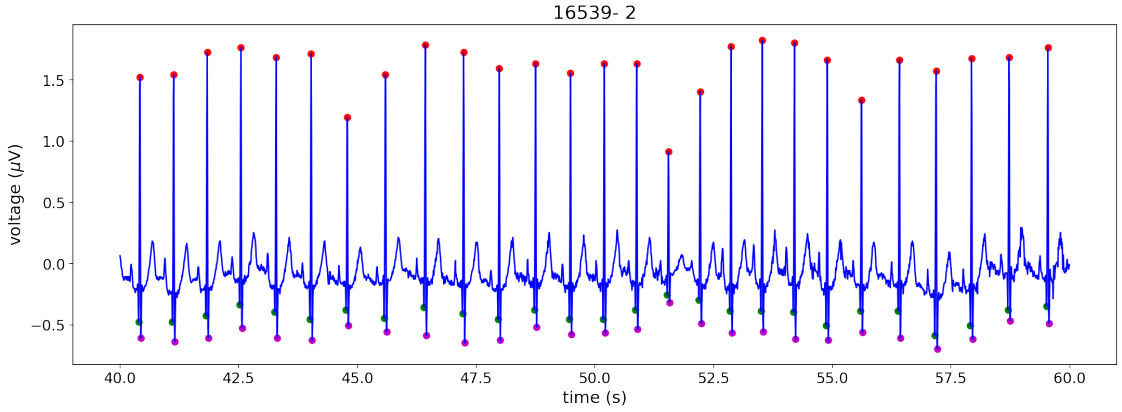

16483- 2
voltage (μV)
time (s)

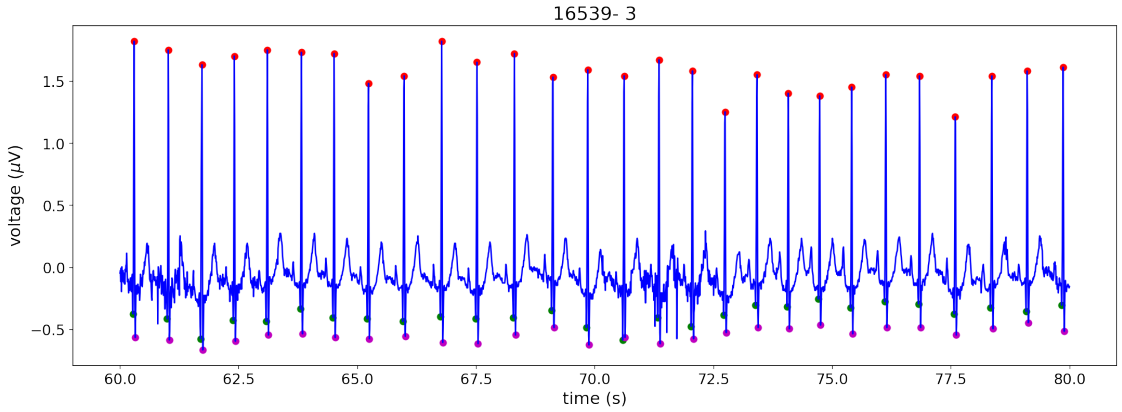

16483- 3
voltage (μV)
time (s)

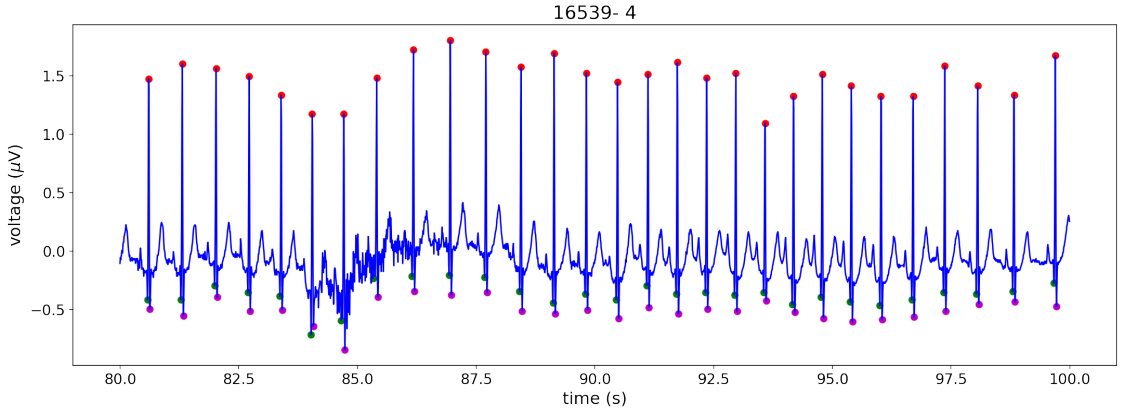

16483- 4
voltage (μV)
time (s)

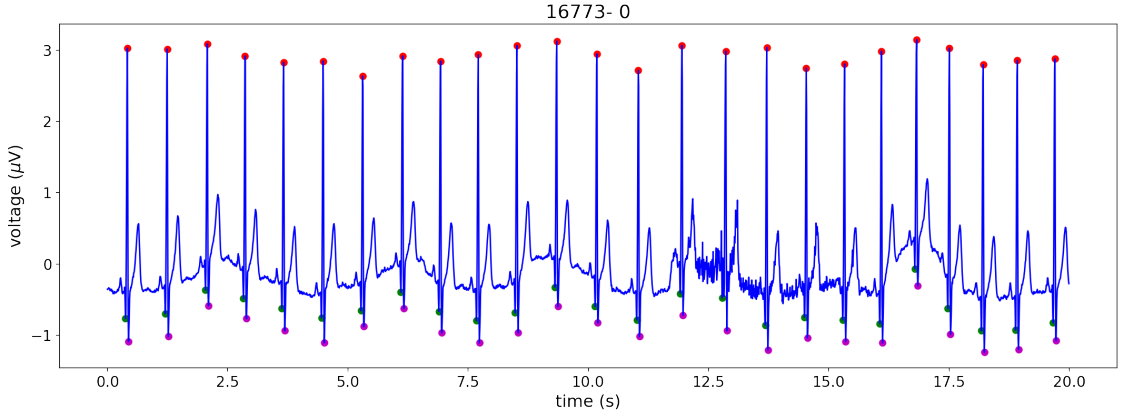

16539- 0
voltage (μV)
time (s)

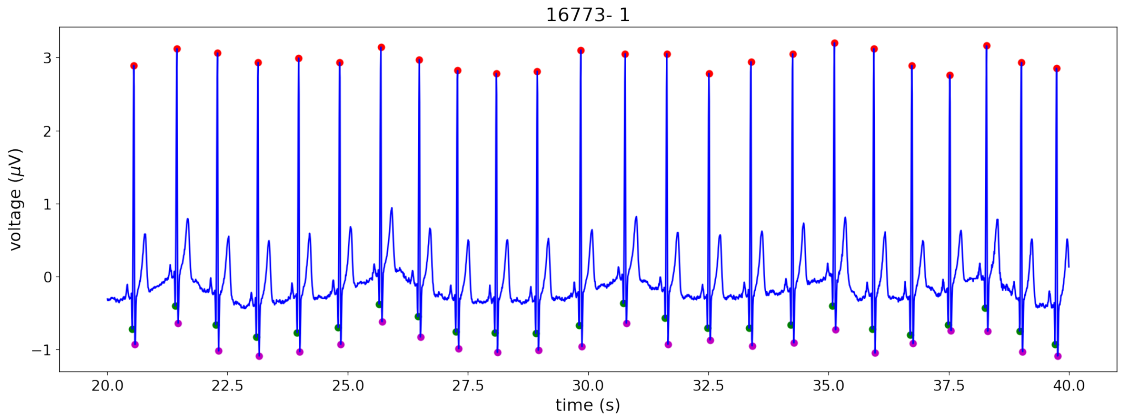

16539- 1
voltage (μV)
1.5
1.0
0.5
0.0
−0.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
time (s)

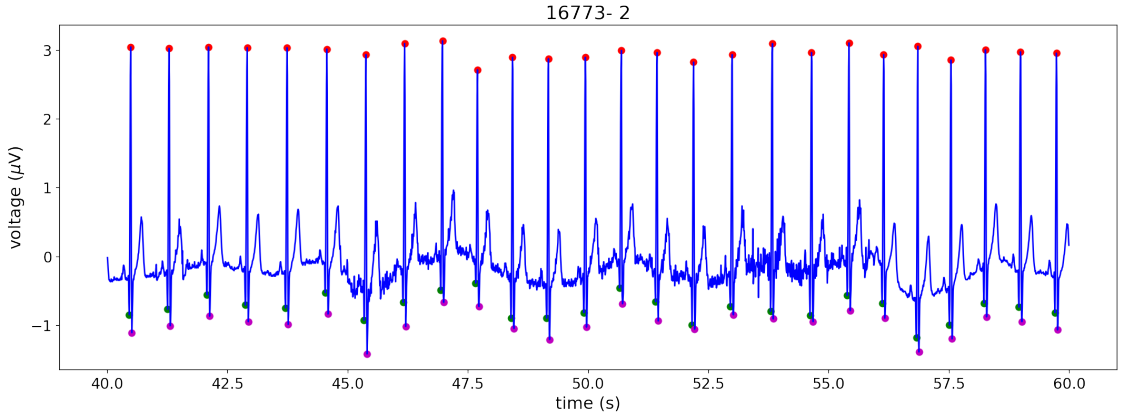

16539- 2
voltage (μV)
1.5
1.0
0.5
0.0
−0.5
40.0
42.5
45.0
47.5
50.0
52.5
55.0
57.5
60.0
time (s)

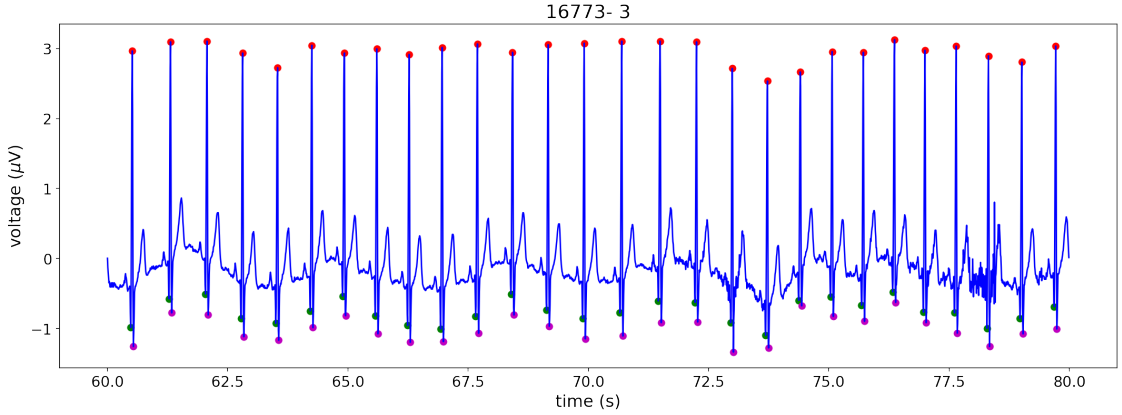

16539- 3
voltage (μV)
1.5
1.0
0.5
0.0
−0.5
60.0
62.5
65.0
67.5
70.0
72.5
75.0
77.5
80.0
time (s)

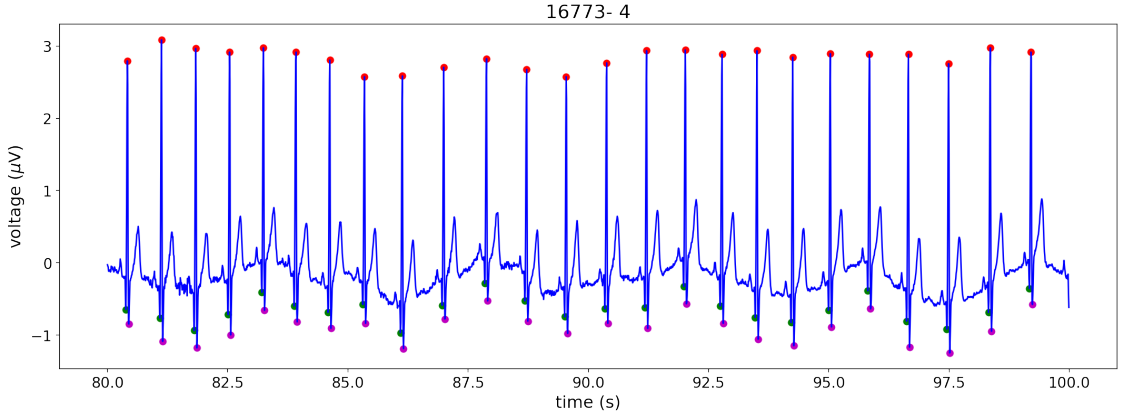

16539- 4
voltage (μV)
time (s)

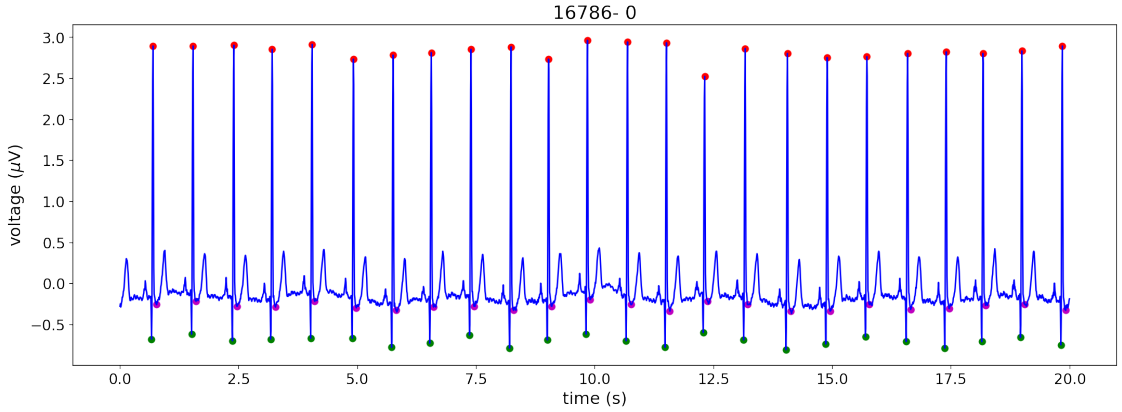

16773- 0
voltage (μV)
time (s)

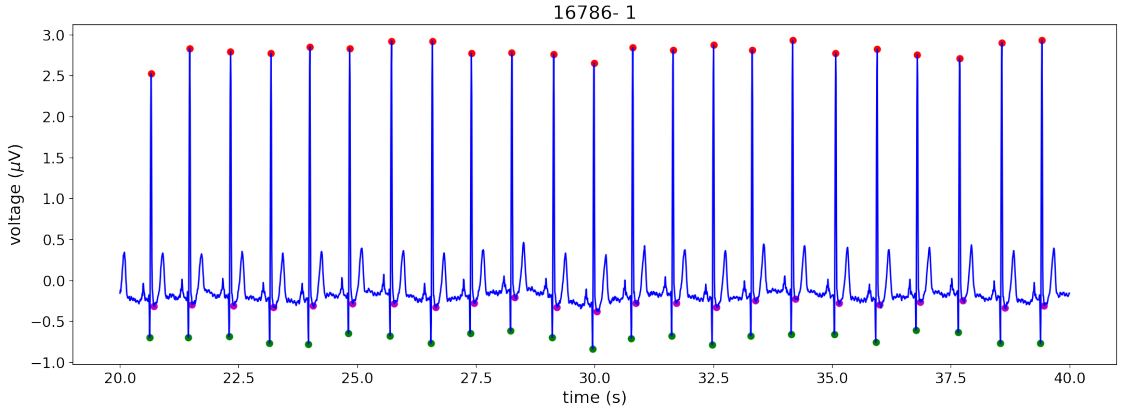

16773- 1
voltage (μV)
time (s)

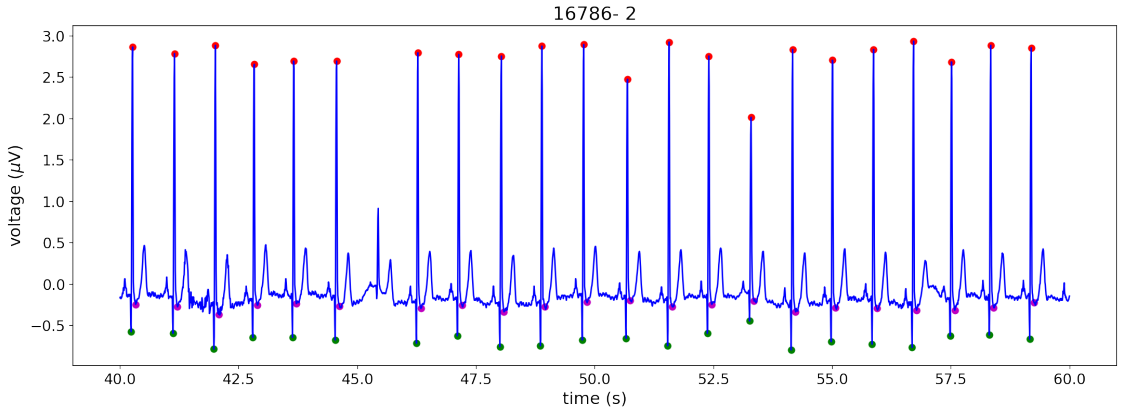

16773- 2
voltage (μV)
time (s)

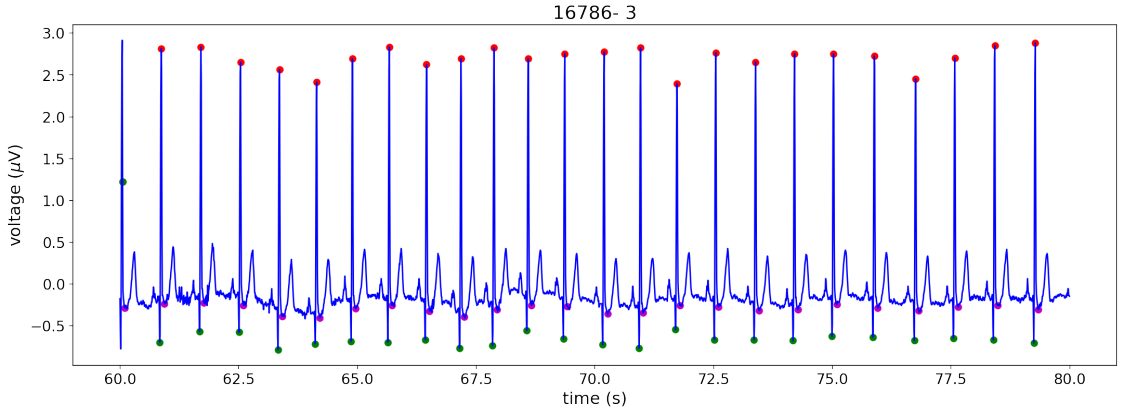

16773- 3
voltage (μV)
time (s)

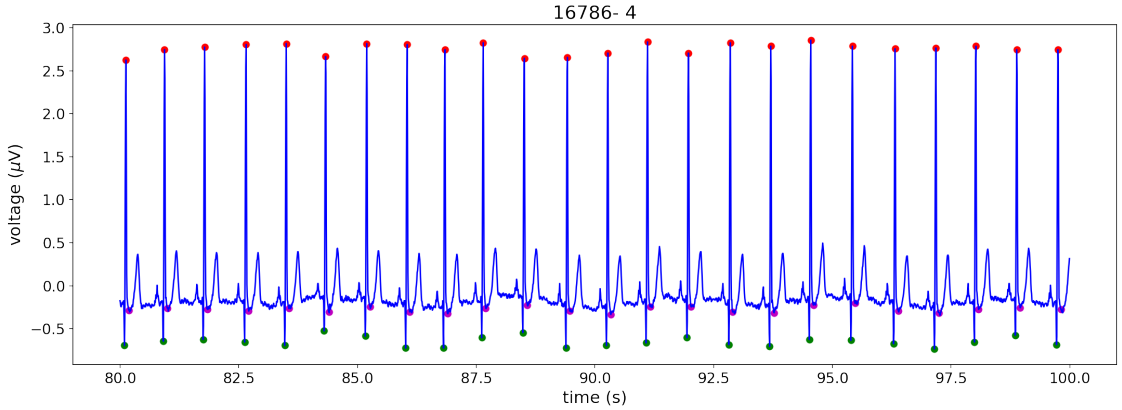

16773- 4
voltage (μV)
time (s)

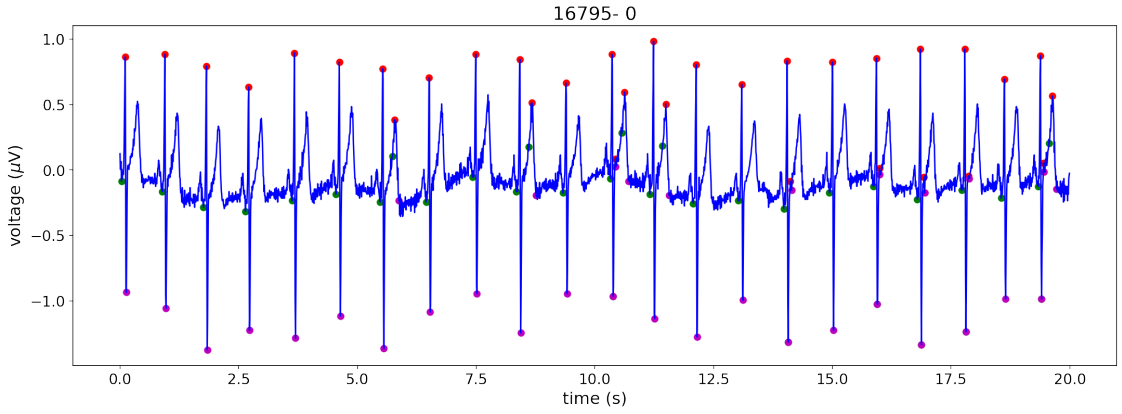

16786- 0
voltage (μV)
time (s)

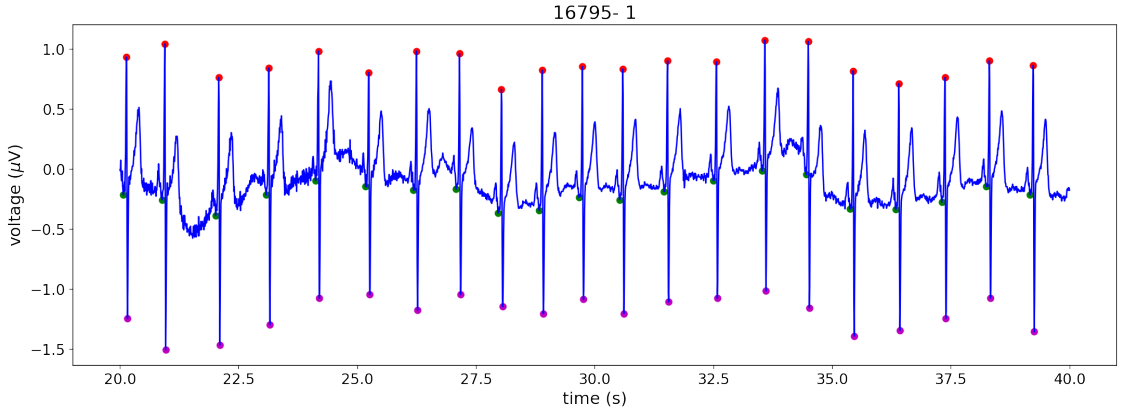

16786- 1
voltage (μV)
time (s)

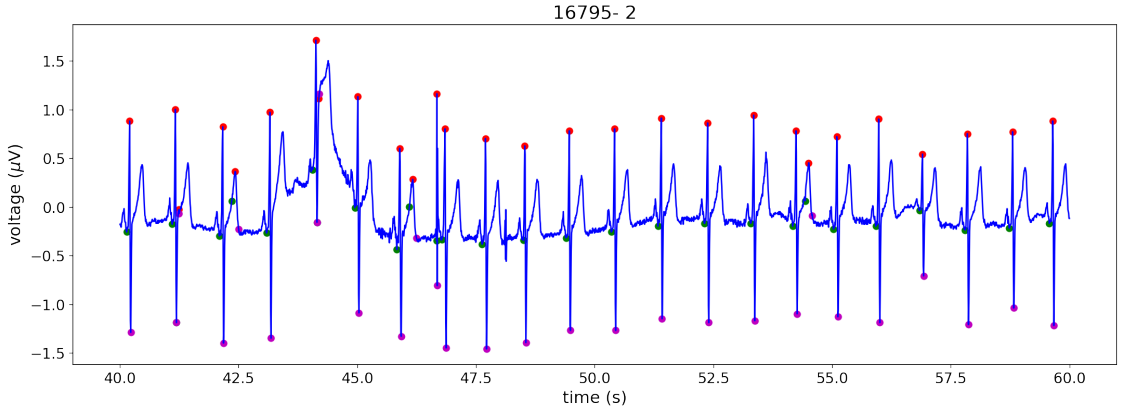

16786- 2
voltage (μV)
time (s)

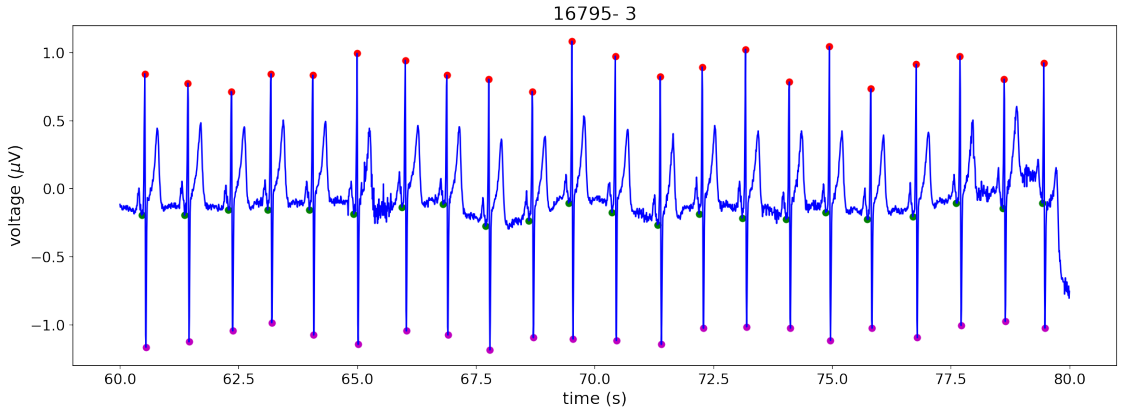

16786- 3
voltage (μV)
time (s)

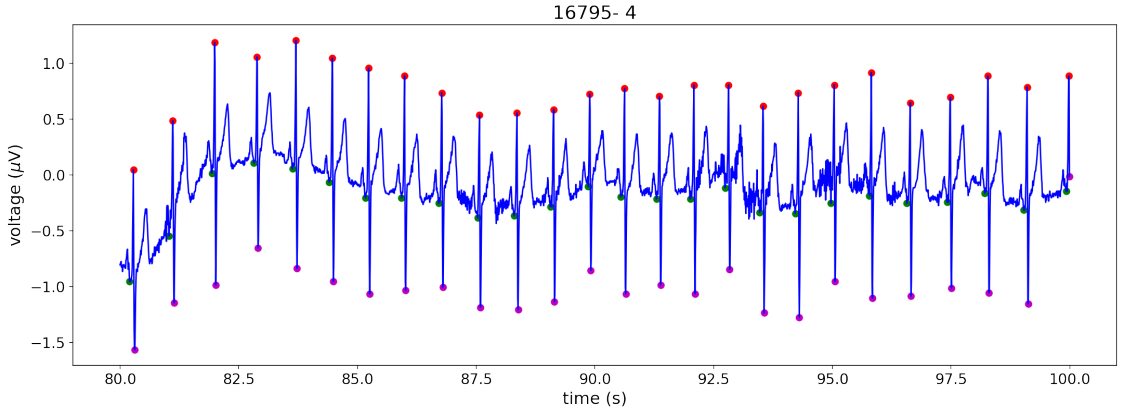

16786- 4
voltage (μV)
time (s)

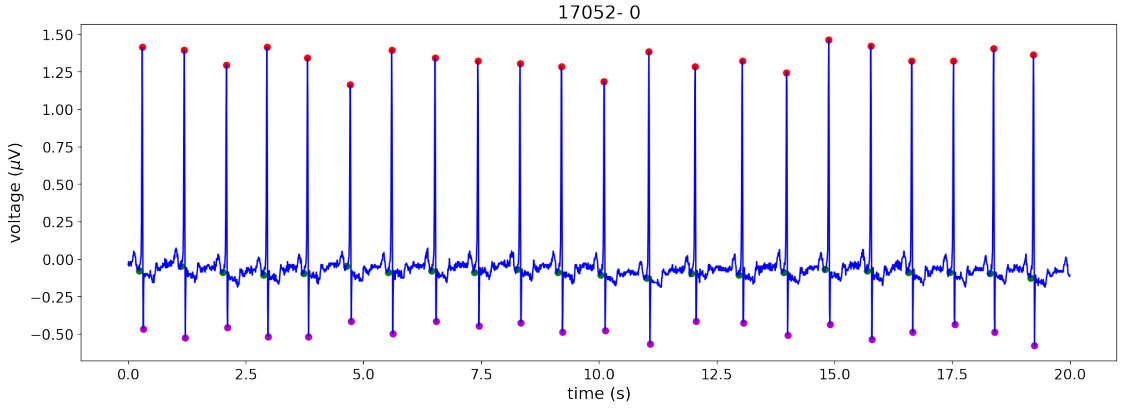

16795- 0
voltage (μV)
time (s)

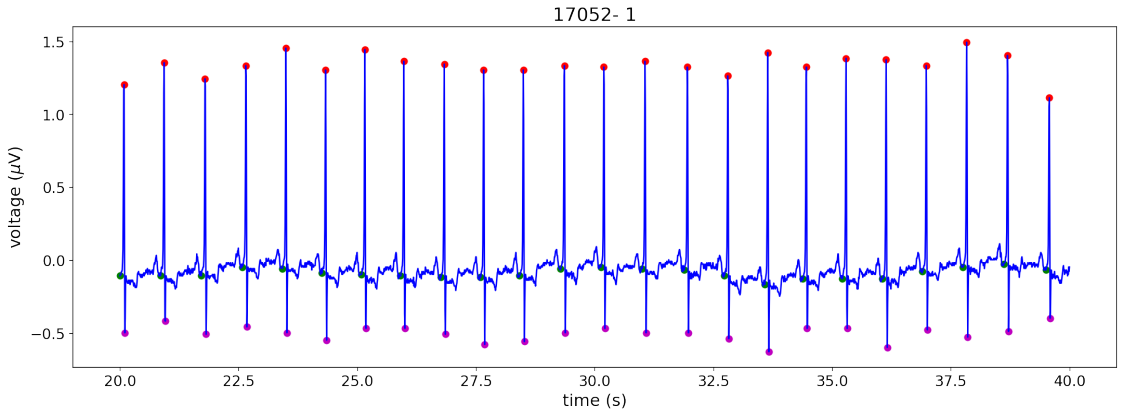

16795- 1
voltage (μV)
time (s)

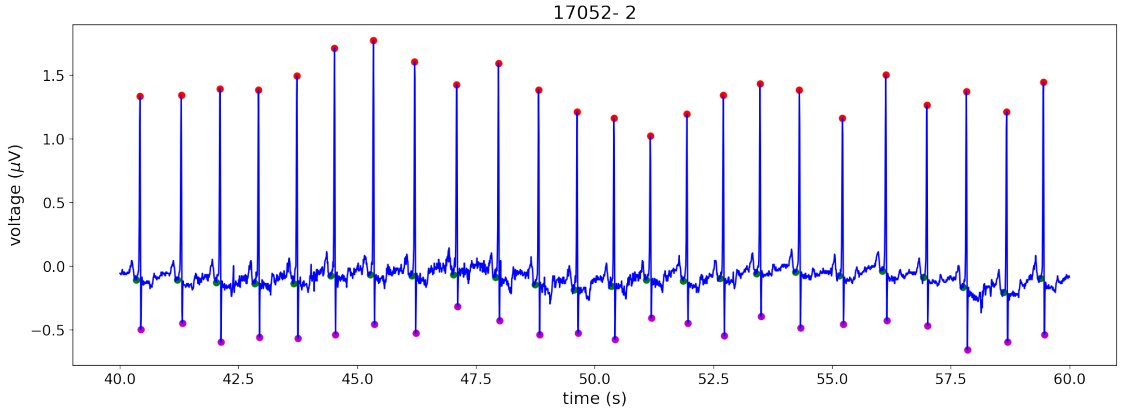

16795- 2
voltage (μV)
time (s)

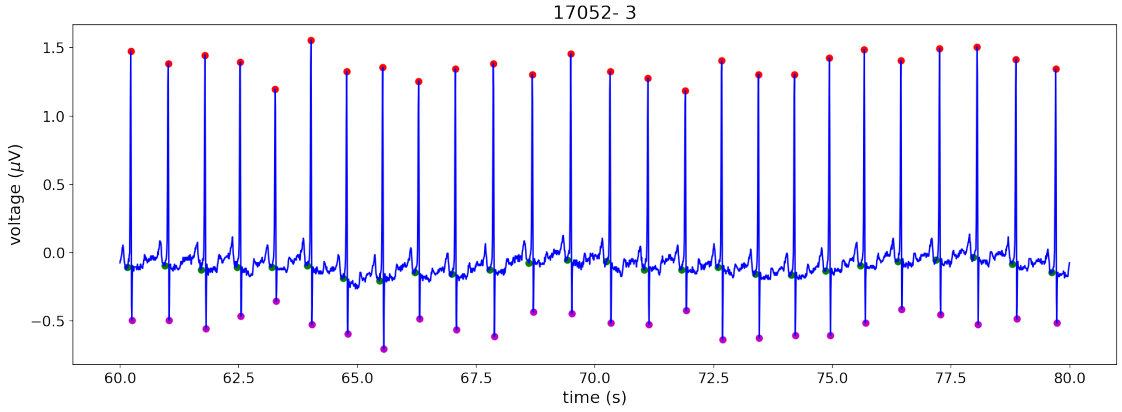

16795- 3
voltage (μV)
time (s)

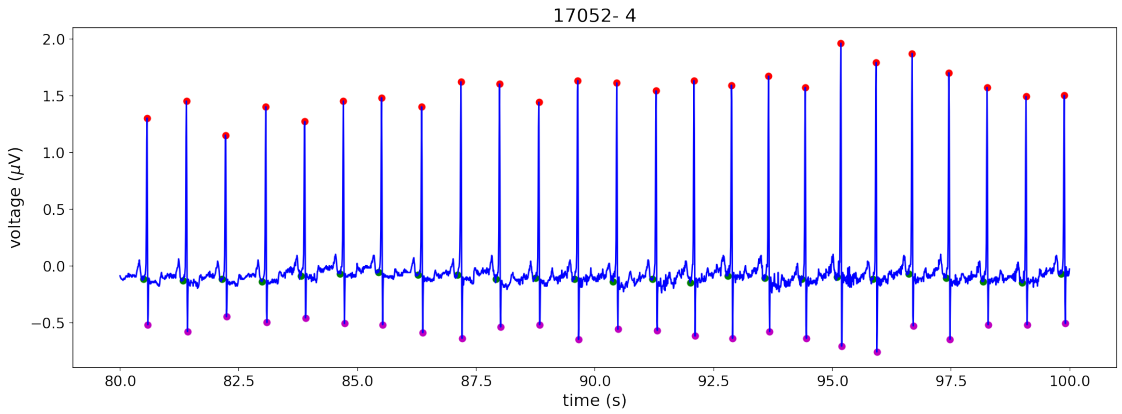

16795- 4
voltage (μV)
time (s)

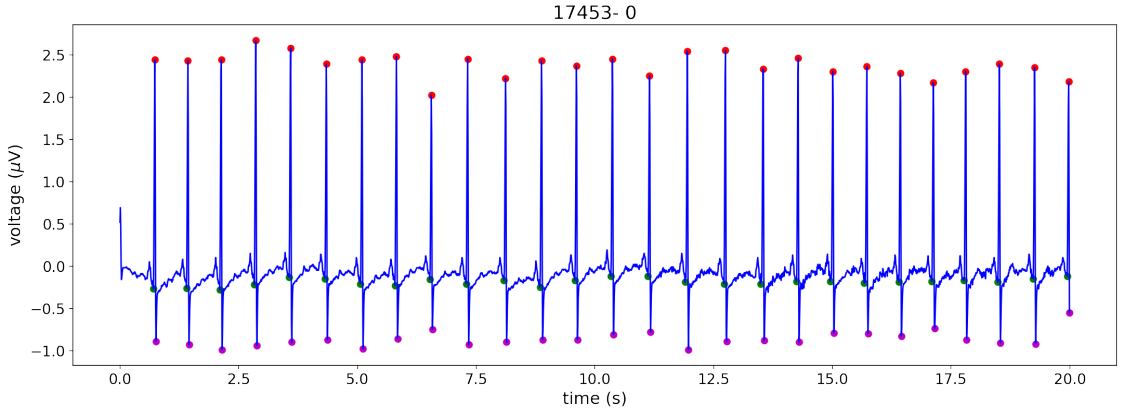

17052- 0
voltage (μV)
time (s)

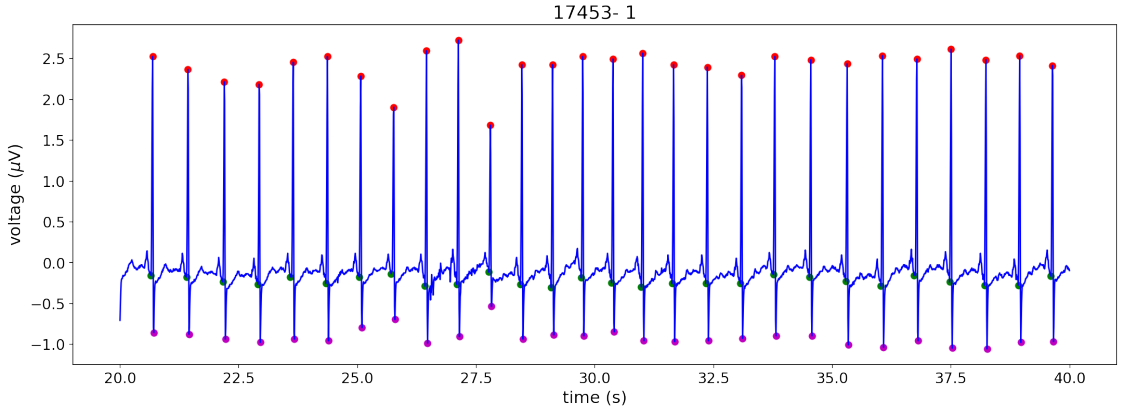

17052- 1
voltage (μV)
time (s)

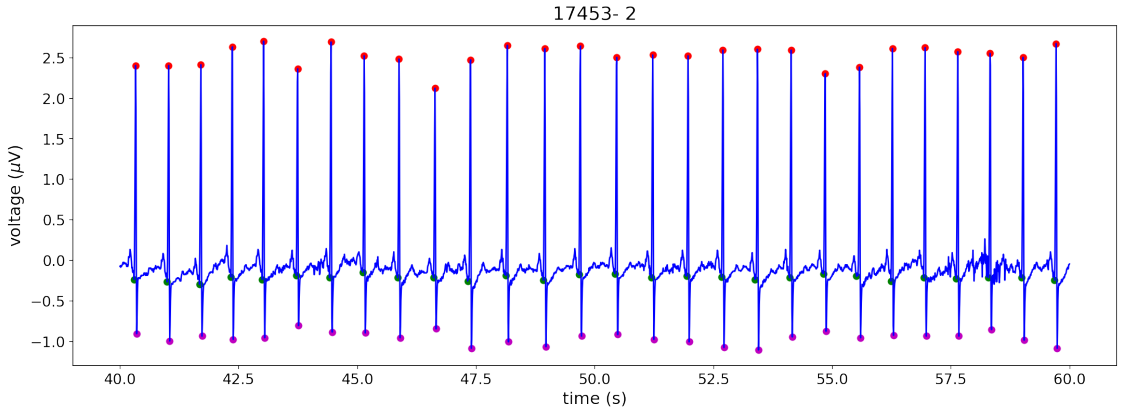

17052- 2
voltage (μV)
1.5
1.0
0.5
0.0
−0.5
40.0
42.5
45.0
47.5
50.0
52.5
55.0
57.5
60.0
time (s)

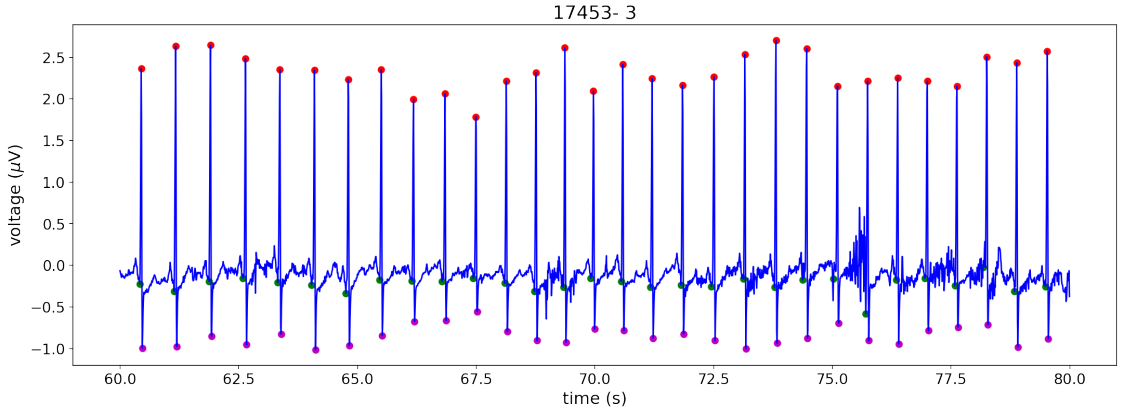

17052- 3
voltage (μV)
1.5
1.0
0.5
0.0
−0.5
60.0
62.5
65.0
67.5
70.0
72.5
75.0
77.5
80.0
time (s)

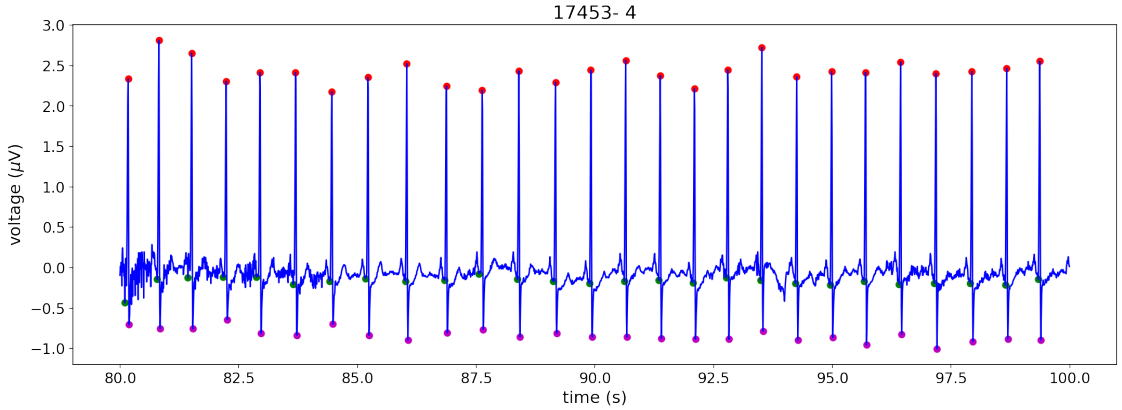

17052- 4
voltage (μV)
2.0
1.5
1.0
0.5
0.0
−0.5
80.0
82.5
85.0
87.5
90.0
92.5
95.0
97.5
100.0
time (s)

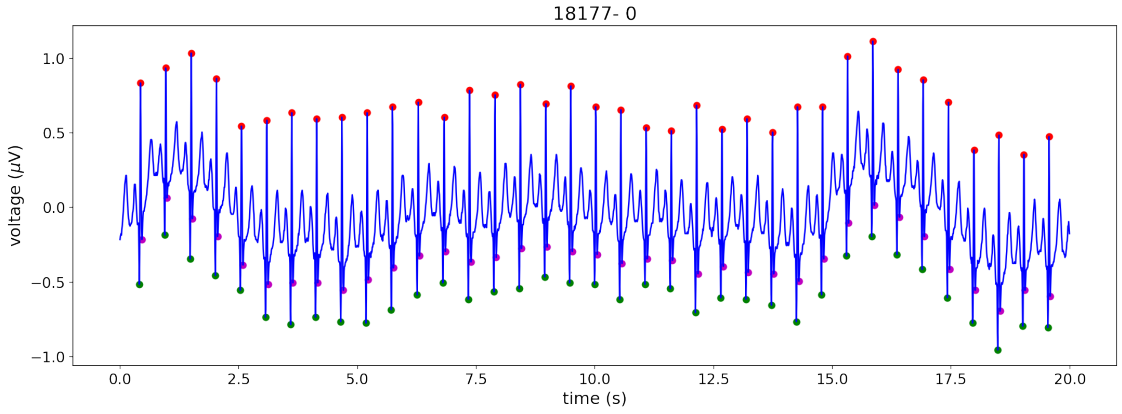

17453- 0
voltage (μV)
time (s)

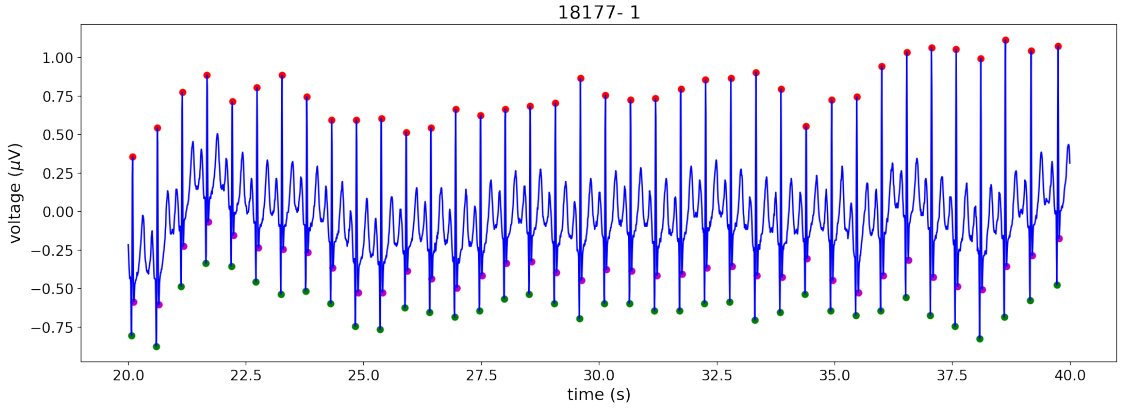

17453- 1
voltage (μV)
time (s)

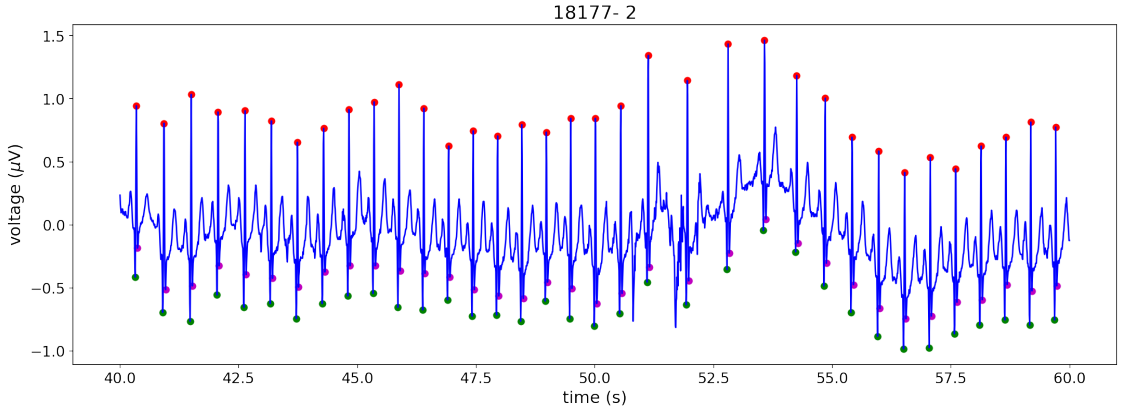

17453- 2
voltage (μV)
time (s)

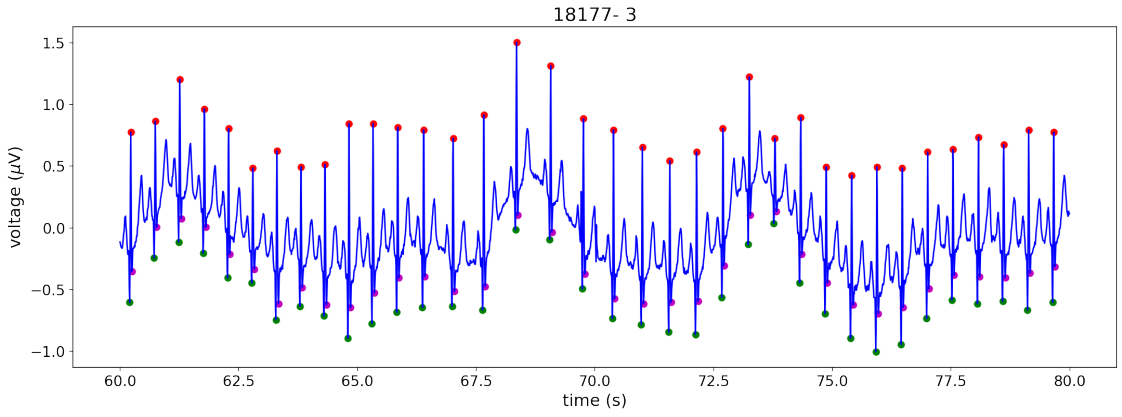

17453- 3
voltage (μV)
time (s)

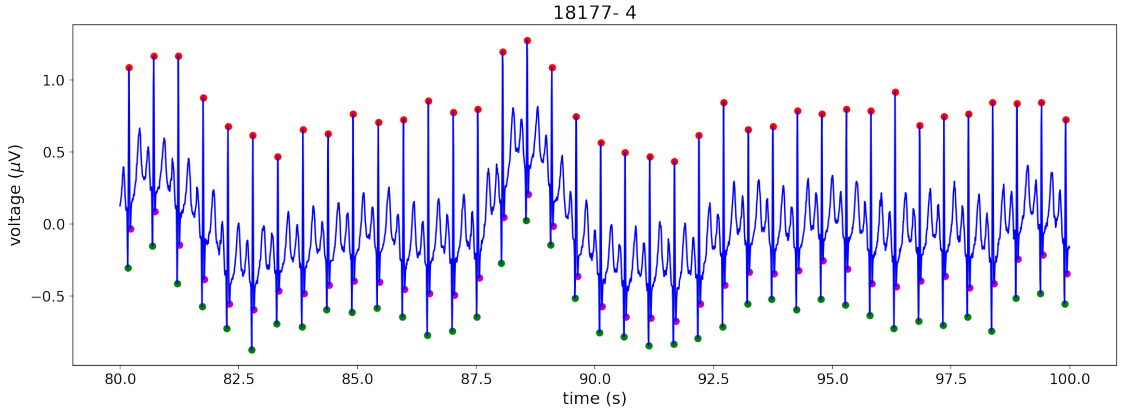

17453- 4
voltage (μV)
time (s)

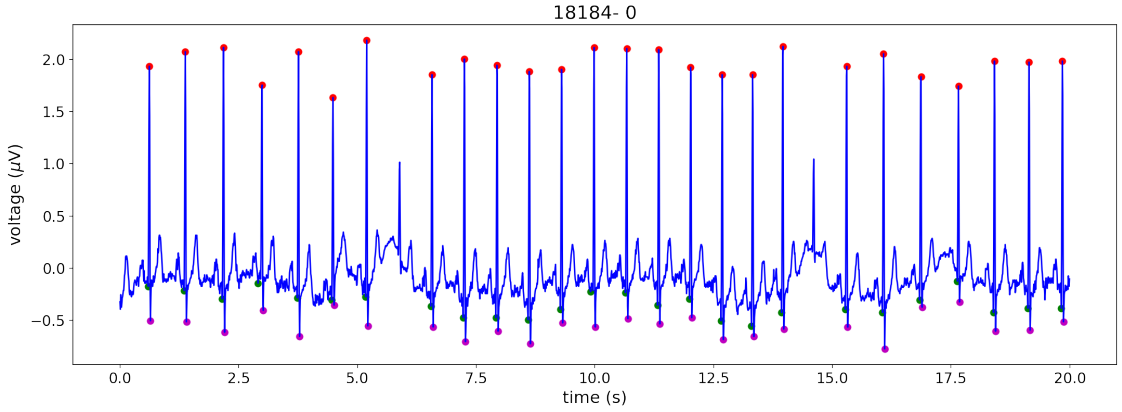

18177- 0
voltage (μV)
time (s)

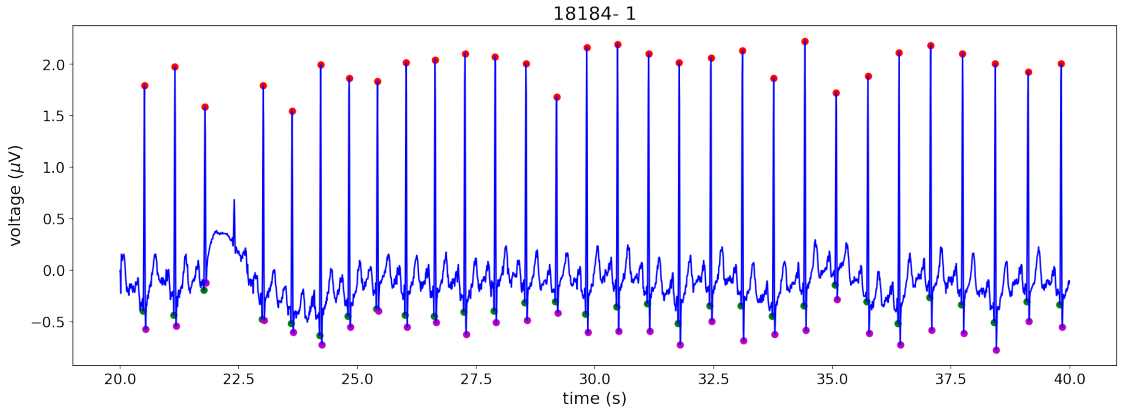

18177- 1
voltage (μV)
time (s)

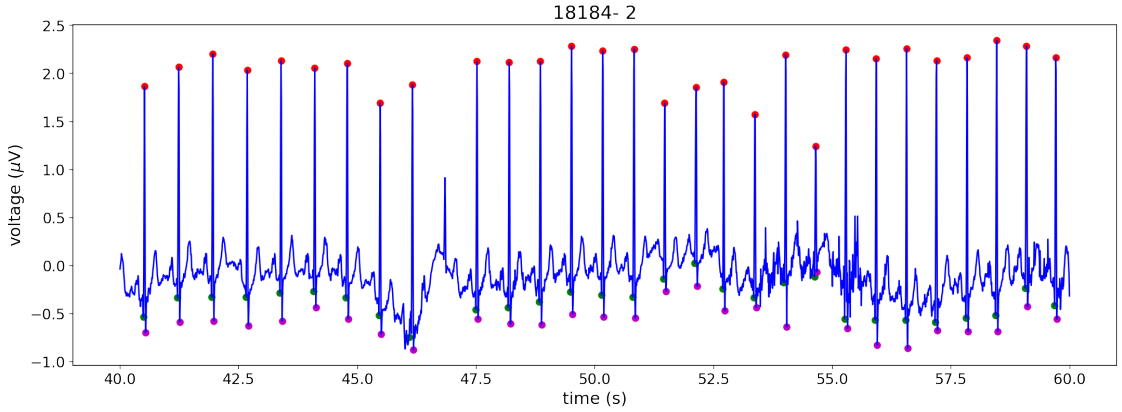

18177- 2
voltage (μV)
time (s)

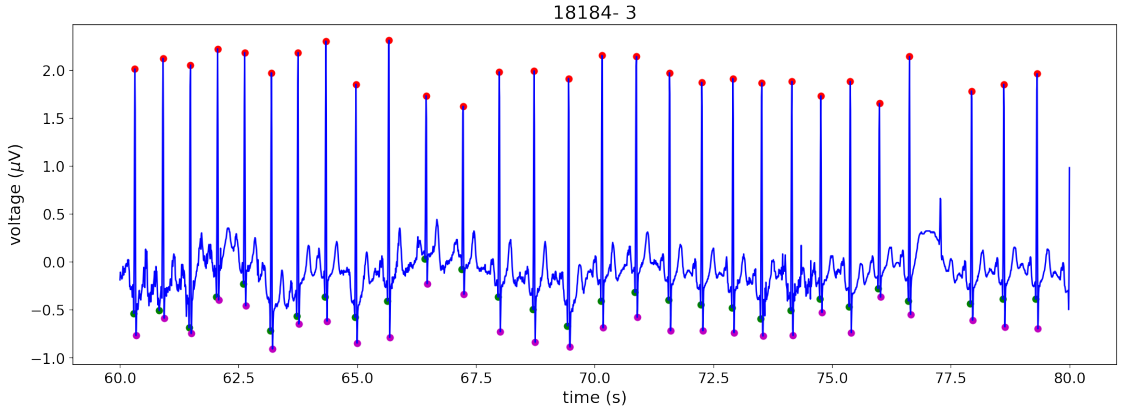

18177- 3
voltage (μV)
time (s)

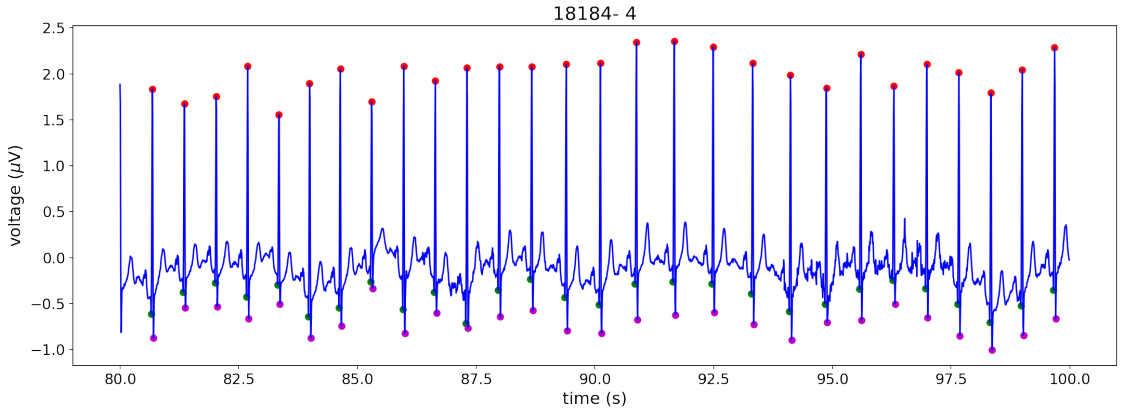

18177- 4
voltage (μV)
1.0
0.5
0.0
−0.5
80.0
82.5
85.0
87.5
90.0
92.5
95.0
97.5
100.0
time (s)

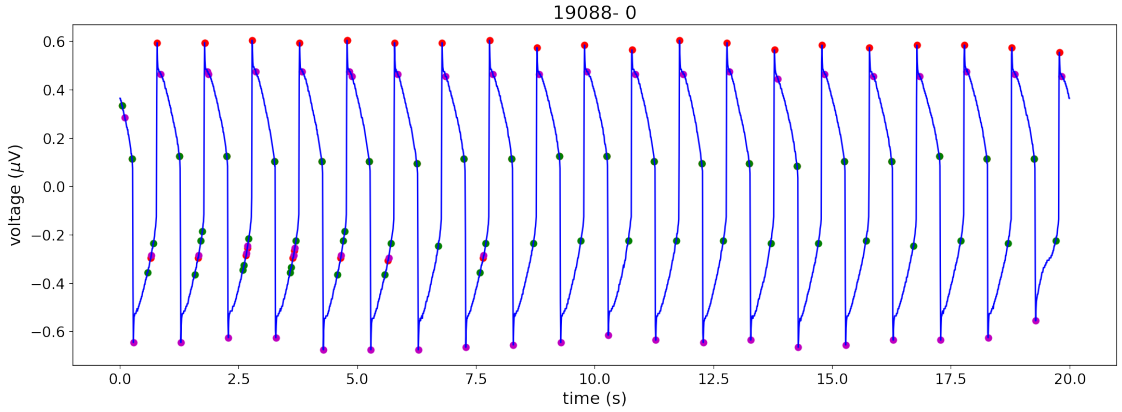

18184- 0
voltage (μV)
2.0
1.5
1.0
0.5
0.0
−0.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
time (s)


18184- 1
voltage (μV)
2.0
1.5
1.0
0.5
0.0
−0.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
time (s)

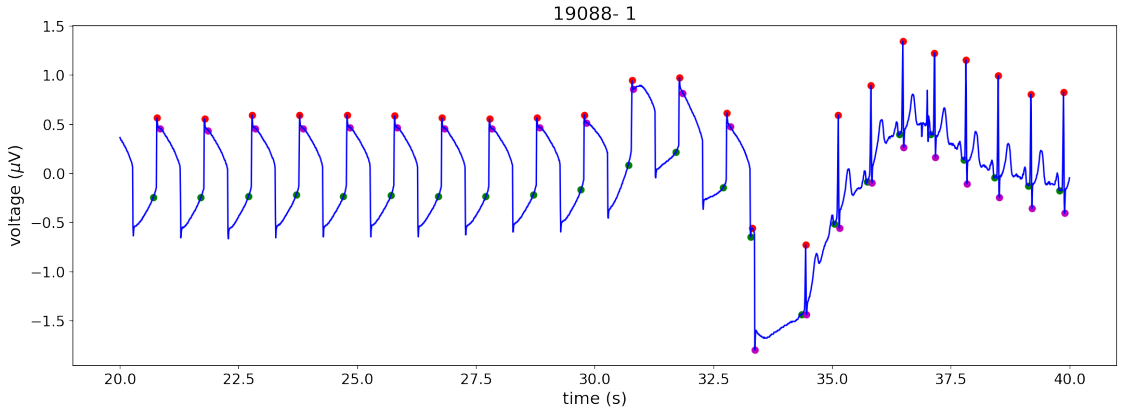

18184- 2
voltage (μV)
time (s)

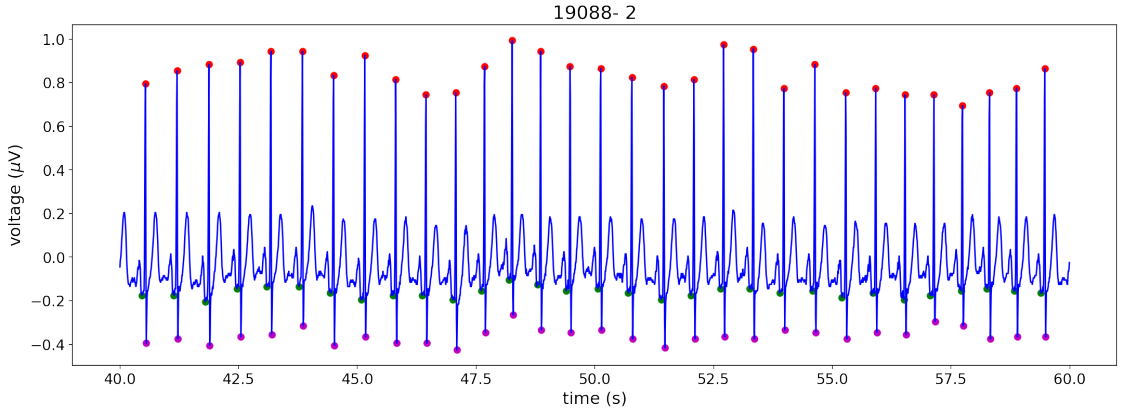

18184- 3
voltage (μV)
time (s)

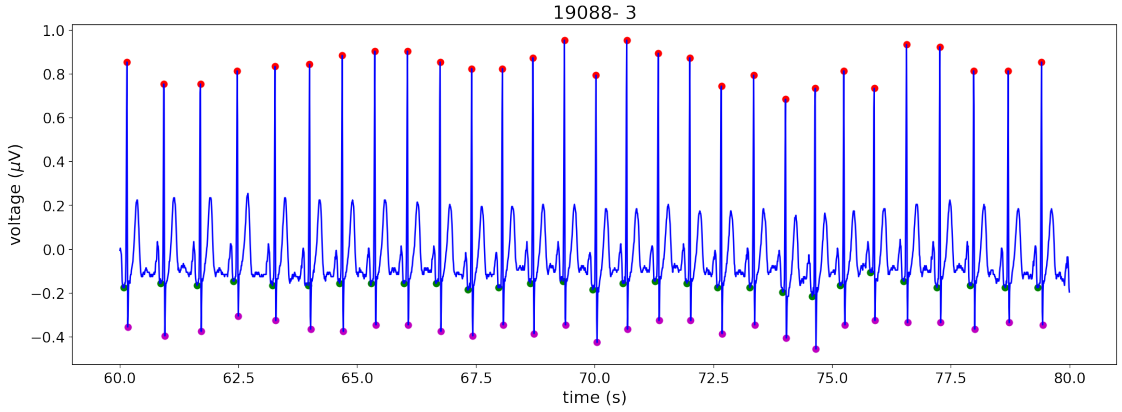

18184- 4
voltage (μV)
time (s)

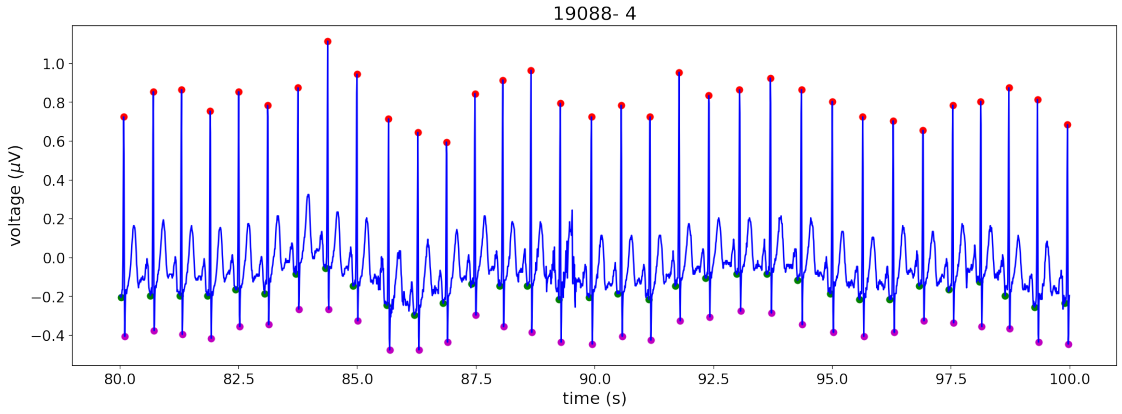


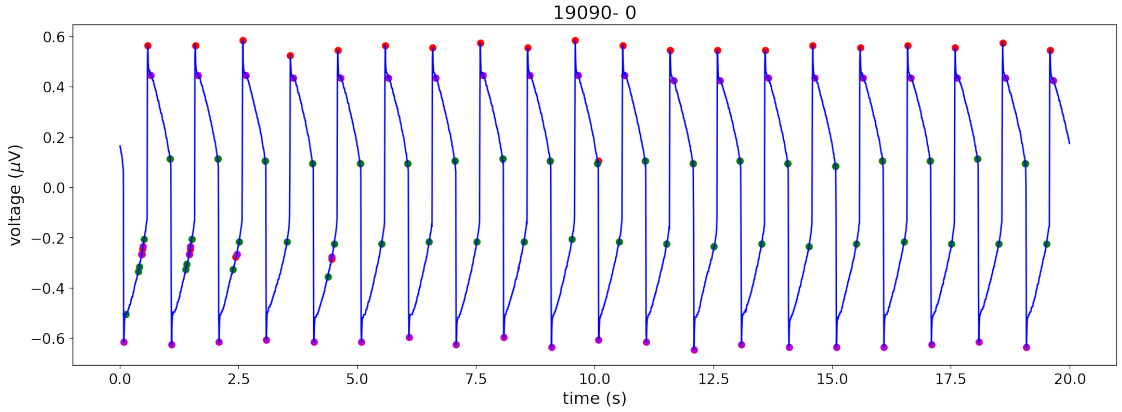

```
/Users/javierserranomolina/miniconda3/envs/IA/lib/python3.7/site-
packages/ipykernel_launcher.py:42: RuntimeWarning: divide by zero encountered in
double_scalars
/Users/javierserranomolina/miniconda3/envs/IA/lib/python3.7/site-
packages/numpy/core/_methods.py:202: RuntimeWarning: invalid value encountered
in subtract
  x = asanyarray(arr - arrmean)
```

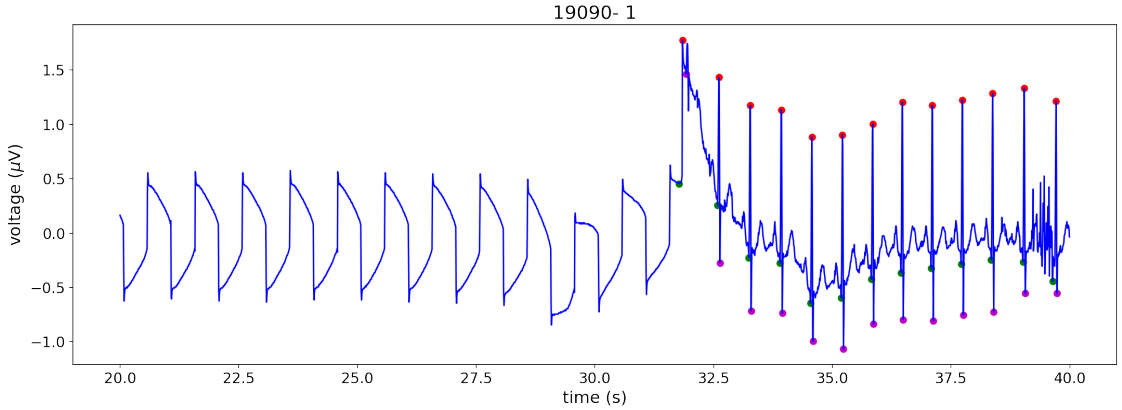


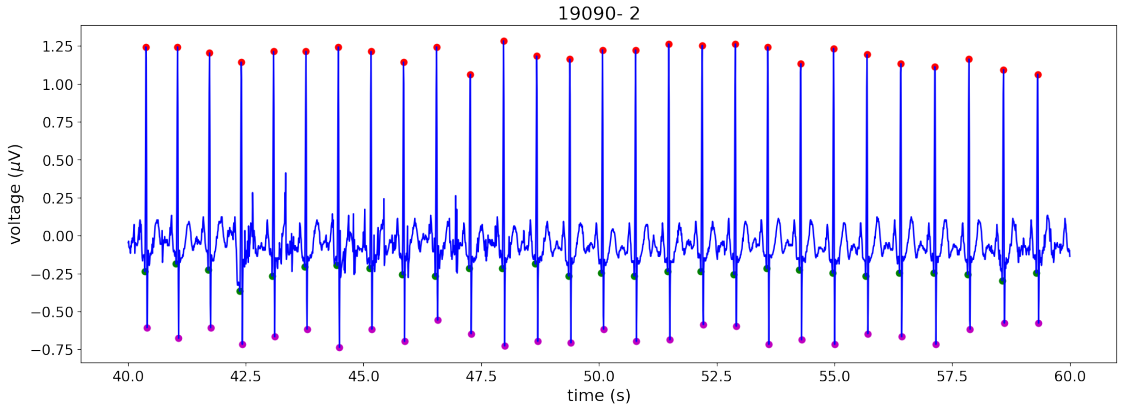

19088- 2
voltage (μV)
time (s)

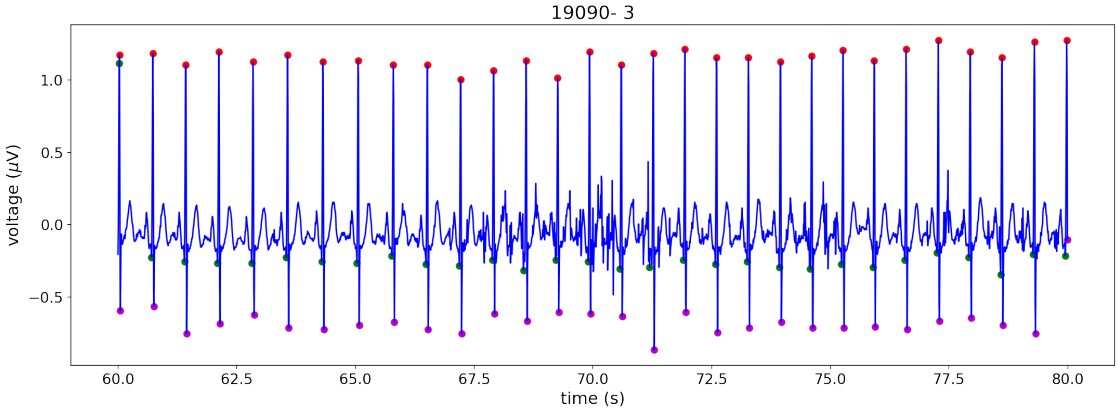

19088- 3
voltage (μV)
time (s)

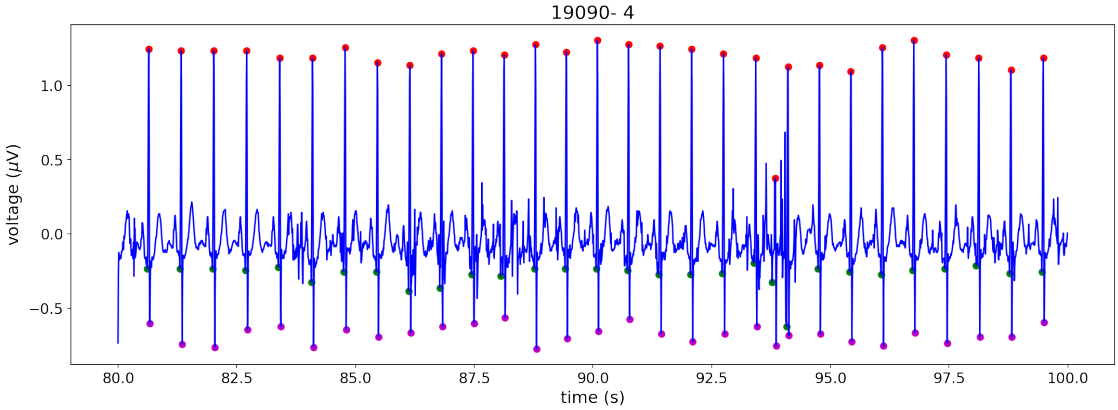

19088- 4
voltage (μV)
time (s)

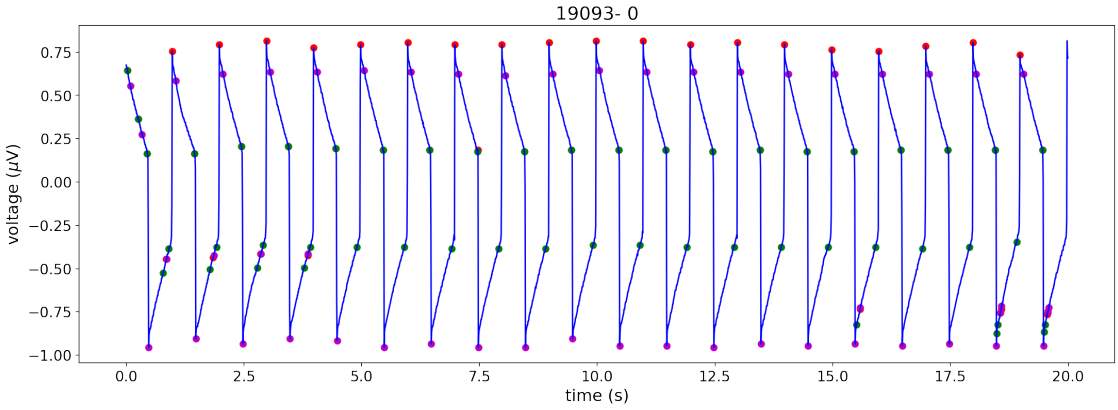

19090- 0
voltage (μV)
time (s)

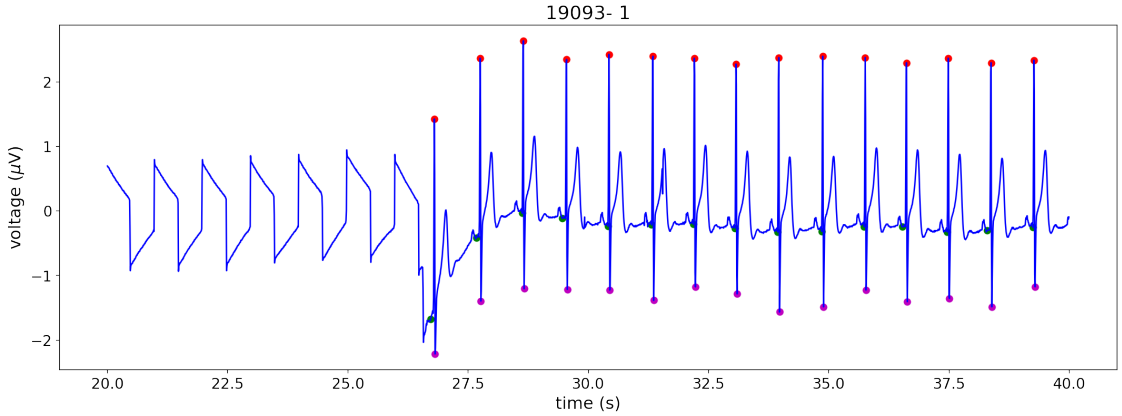

19090- 1
voltage (μV)
time (s)

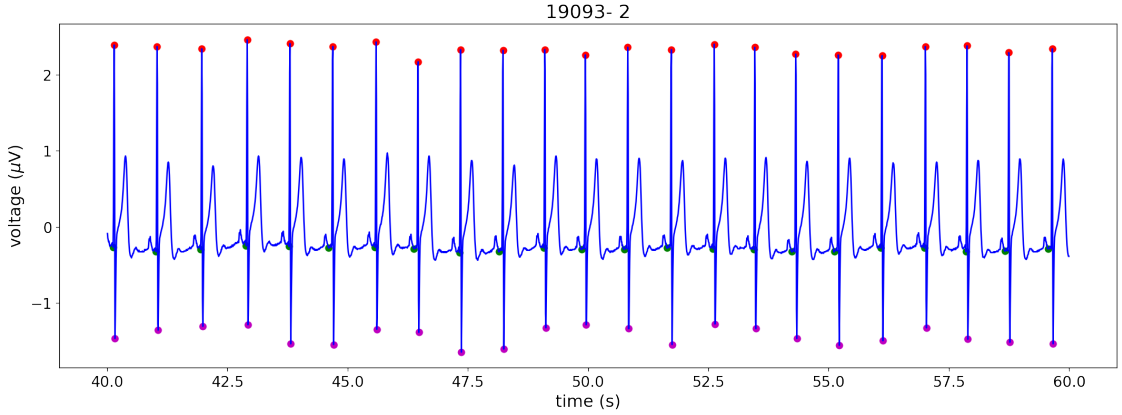

19090- 2
voltage (μV)
time (s)

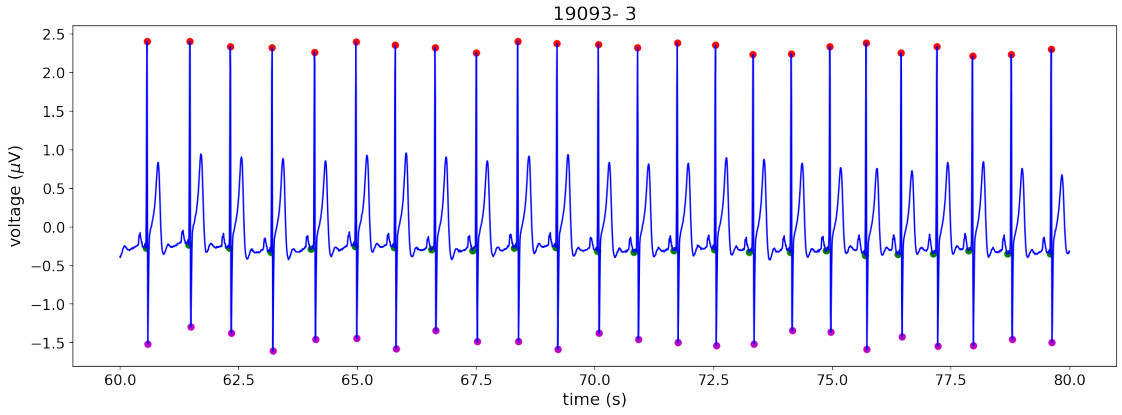

19090- 3
voltage (μV)
1.0
0.5
0.0
−0.5
60.0
62.5
65.0
67.5
70.0
72.5
75.0
77.5
80.0
time (s)

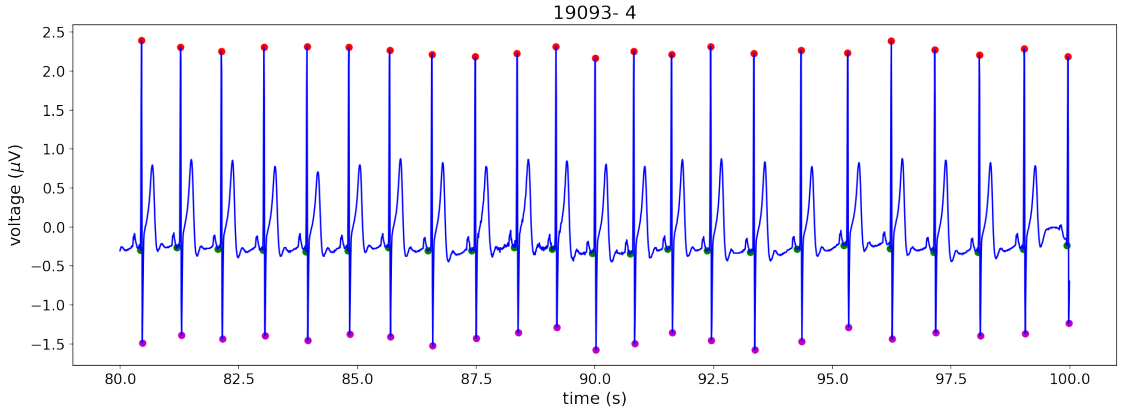

19090- 4
voltage (μV)
1.0
0.5
0.0
−0.5
80.0
82.5
85.0
87.5
90.0
92.5
95.0
97.5
100.0
time (s)

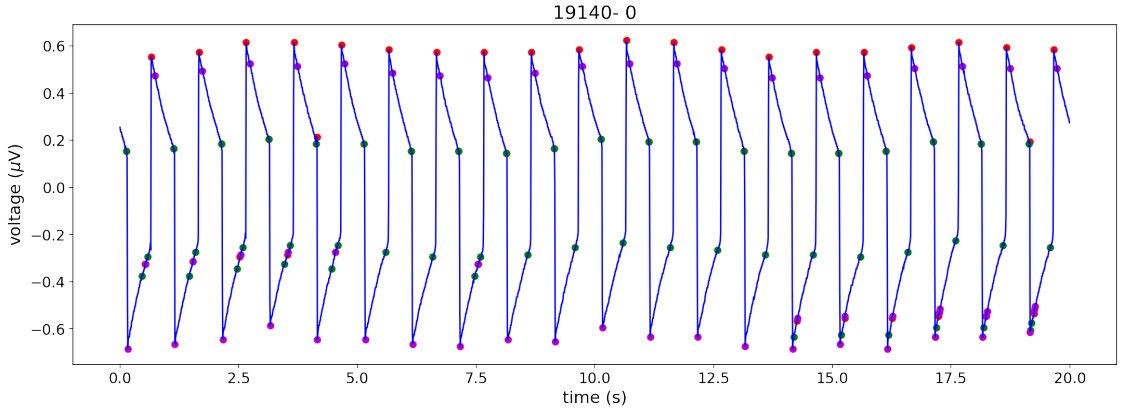

19093- 0
voltage (μV)
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
time (s)

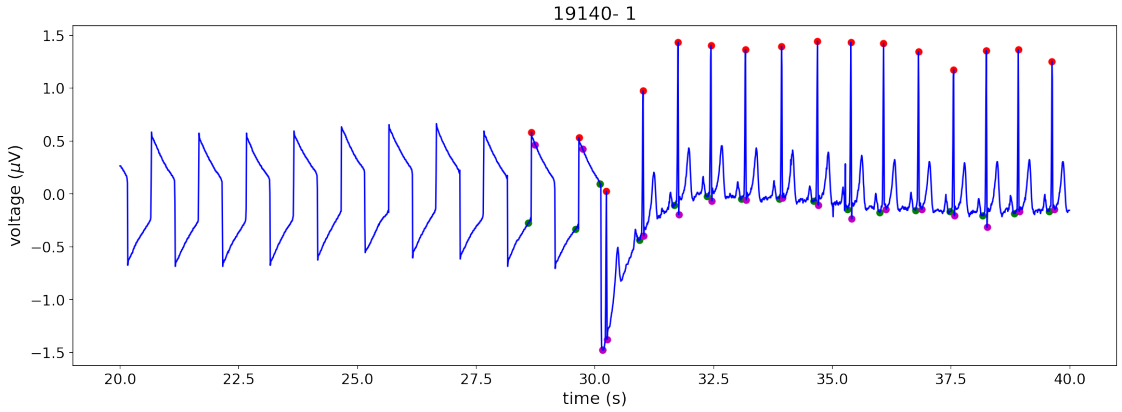

19093- 1
voltage (μV)
time (s)

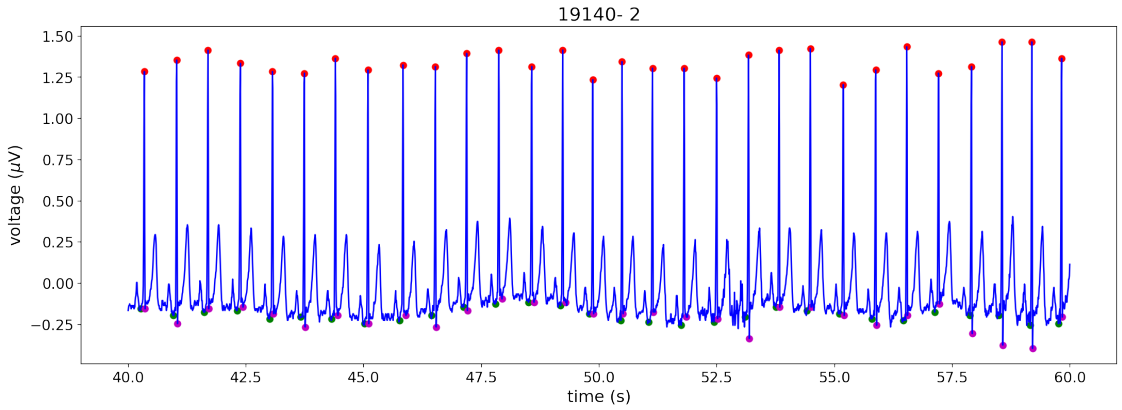

19093- 2
voltage (μV)
time (s)

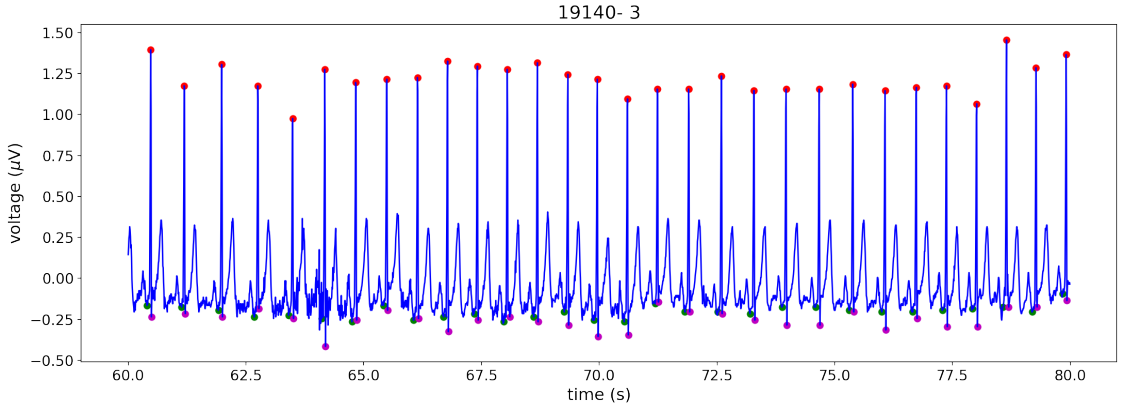

19093- 3
voltage (μV)
time (s)

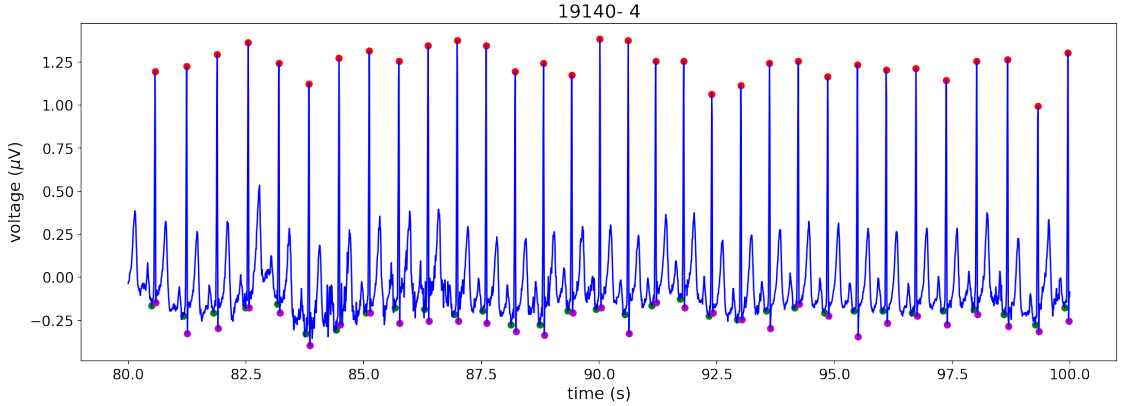

19093- 4
voltage (μV)
time (s)


19140- 0
voltage (μV)
time (s)


19140- 1
voltage (μV)
time (s)







[10]:
```
#Now we print how many samples we have of each group
print(len(rn))
print(len(ra))
```

64
53

[11]:
```
#We define a funtion that compares each variable from both groups in a histogram
def histogram(var, rn=rn,ra=ra):
    labels=["Mean QRS lenght (s)","Standard Deviation of QRS lenght (s)", "Max␣
 ↪QRS lenght (s)", "Min QRS lenght (s)","Mean heart rate (s)","Standard␣
 ↪deviation of heart rate (s)", "Max heart rate (s)", "Min heart rate (s)"]
    plt.figure(figsize=(18,6))
    plt.hist(rn[:,var],color="b", label="Normal", bins=30,density=True,alpha=0.
 ↪5,histtype='stepfilled')
    plt.hist(ra[:,var],color="r", label="Arrythmia", bins=30,alpha=0.
 ↪5,histtype='stepfilled', density=True)
    plt.legend()
    plt.title(labels[var],color="white")
    plt.tick_params(axis='x', colors='white')
    plt.tick_params(axis='y', colors='white')
    plt.savefig("hist_"+str(var)+".png")
    plt.show;
```

[12]:
```
#We define a function that graphs in a scatter plot any two variables and␣
 ↪compares them
def compare_var(var1,var2,rn=rn,ra=ra):
    labels=["Mean QRS lenght (s)","Standard Deviation of QRS lenght (s)", "Max␣
 ↪QRS lenght (s)", "Min QRS lenght (s)","Mean heart rate (s)","Standard␣
 ↪deviation of heart rate (s)", "Max heart rate (s)", "Min heart rate (s)"]
    plt.figure(figsize=(18,6))
    plt.scatter(rn[:,var1],rn[:,var2],color='b',label="Normal")
    plt.scatter(ra[:,var1],ra[:,var2],color='r',label="Arrhythmia")
    plt.legend()
    plt.xlabel(labels[var1],color="white")
    plt.ylabel(labels[var2],color="white")
    plt.title(labels[var1]+" vs "+labels[var2],color="white")
    plt.tick_params(axis='x', colors='white')
    plt.tick_params(axis='y', colors='white')
    plt.savefig("comp_"+str(var1)+"_"+str(var2)+".png")
    plt.show;
```

#A histogram is made for every variable for i in range(0,7): histogram(i)

#A variable comparison is made for every combination of two variables. for a in range(0,7): for i in range(0,7): if i!=a: compare_var(a,i)

1.4.1 Observations of the variables

Notice that for almost every variable there is a slight diference among the two groups except for one. Can you guess which one from the graphs?

1.5 Neural Network

First we will do some pre-processing of the obtained data. First we will need to mark each sample. A zero or a one will be appended to the array of variables, depending on whether it corresponds to a normal patient or a patient with arrythmia respectively. Then we need to shuffle them. If we don't shuffle our data, the samples from one patient will be together. This might result in a biased training.

[13]:
```
control=np.append(rn,np.zeros([len(rn),1]),axis=1)
```

[14]:
```
arrythm=np.append(ra,np.ones([len(ra),1]),axis=1)
```

[15]:
```
np.random.shuffle(control)
np.random.shuffle(arrythm)
```

[16]:
```
print(np.shape(control))
print(np.shape(arrythm))
```

```
(64, 8)
(53, 8)
```

1.5.1 Dividing training group vs test group

Now we need to divide all of the data in train and test. The train sample will be used to otimize our network. The validation group will be used to evaluate the accuracy of the trained network. Why do you think we can't evaluate an algorithm with the same data we used to train it?

[17]:
```
#The two sets are created with equal parts of normal samples and arrythmia␣
↪samples.
train=np.append(control[:45],arrythm[:45],axis=0)
test=np.append(control[45:len(arrythm)],arrythm[45:],axis=0)
print(np.shape(train))
```

```
(90, 8)
```

[18]:
```
#Then the training samples are shuffled to avoid a bias in the trainig.
np.random.shuffle(train)
```

[19]:
```
#The variables are redivided.
#"x" corresponds to the independent variables
#"y" corresponds to the dependent variable (arrythmia or normal)
train_x=train[:,:-1]
train_y=train[:,-1:]
test_x=test[:,:-1]
test_y=test[:,-1:]
```

[20]:
```
print(train_x.shape)
print(train_y.shape)
print(test_x.shape)
```

```
print(test_y.shape)
```

```
(90, 7)
(90, 1)
(16, 7)
(16, 1)
```

[21]:
```
#Now we import the modules needed to create the neural network
import pandas as pd
import seaborn as sn
```

[22]:
```
from keras.models import Sequential
from keras.layers import Input, Dense, Flatten, Dropout
from keras.layers import Activation
from keras.optimizers import SGD
from keras.models import Model
from keras.utils import plot_model
from keras import initializers
from keras import optimizers
import tensorflow as tf
```

```
Using TensorFlow backend.
```

[23]:
```
import networkx as nx

#We define a set of funtions needed to visualize the architechture of the␣
 ↪network
class Network(object):

    def  __init__ (self,sizes):
        self.num_layers = len(sizes)
        print("It has", self.num_layers, "layers,")
        self.sizes = sizes
        print("with the following number of nodes per layer",self.sizes)
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
        self.weights = [np.random.randn(y, x)
                        for x, y in zip(sizes[:-1], sizes[1:])]

    def feedforward(self, x_of_sample):
        """Return the output of the network F(x_of_sample) """
        for b, w in zip(self.biases, self.weights):
            x_of_sample = sigmoid(np.dot(w, x_of_sample)+b)
        return x_of_sample

    def graph(self,sizes):
        a=[]
        ps={}
```

```
        Q = nx.Graph()
        for i in range(len(sizes)):
            Qi=nx.Graph()
            n=sizes[i]
            nodos=np.arange(n)
            Qi.add_nodes_from(nodos)
            l_i=Qi.nodes
            Q = nx.union(Q, Qi, rename = (None, 'Q%i-'%i))
            if len(l_i)==1:
                ps['Q%i-0'%i]=[i/(len(sizes)), 1/2]
            else:
                for j in range(len(l_i)+1):
                    ps['Q%i-%i'%(i,j)]=[i/(len(sizes)),(1/
 ↪(len(l_i)*len(l_i)))+(j/(len(l_i)))]
            a.insert(i,Qi)
        for i in range(len(a)-1):
            for j in range(len(a[i])):
                for k in range(len(a[i+1])):
                    Q.add_edge('Q%i-%i' %(i,j),'Q%i-%i' %(i+1,k))
        nx.draw(Q, pos = ps)
        plt.savefig("Arq.png")
```

[24]:
```
#We define the numer of neurons in each layer. We cant change the numer of␣
 ↪input or output neurons scince they correspond to the variables we already␣
 ↪have.
n_x=7
n_h1=16
n_h2=8
n_y=1
```

#Graphical representation of the network layers = [n_x,n_h1,n_h2,n_y] net = Network(layers) net.graph(layers)

[25]:
```
#Now we define the functions of each layer
model = Sequential()

input_nodes = n_x     #input layer has n_x nodes
hlayer1_nodes = n_h1   #first hidden layer has n_h1 nodes
hlayer2_nodes = n_h2   #second hidden layer has n_h2 nodes
output_nodes = n_y    #output layer has n_y node


#For the first hidden layer, it is necessary to indicate its input layer, which␣
 ↪corresponds to
#the input layer of the network
```

```
model.add(Dense(hlayer1_nodes,  kernel_initializer='identity',␣
 ↪bias_initializer='zeros', input_dim=input_nodes, activation='tanh'))

model.add(Dense(hlayer2_nodes,  kernel_initializer='identity',␣
 ↪bias_initializer='zeros', input_dim=n_h1, activation='relu'))

#For any other hidden layer its input layer is not indicated. Its input layer␣
 ↪is the hidden layer before it

#The following layer is the last layer of the network. It corresponds to the␣
 ↪output layer of the network
model.add(Dense(output_nodes,activation='sigmoid'))
```

#A mental map of the dimentions of the array in each layer plot_model(model, to_file='model.png', show_shapes=True, rankdir='TB', show_layer_names=True)

model.summary()

[26]:
```
# We define the optimizing function and their hyperparameters: learining␣
 ↪rate(lr) in the present case
model.compile(loss='binary_crossentropy', optimizer=optimizers.adam(lr=0.001),␣
 ↪metrics=['accuracy'])
```

[27]:
```
validation_portion = 0.11
epochs = 500

#The network is trained
history = model.fit(train_x, train_y, epochs=epochs, validation_split =␣
 ↪validation_portion)
```

```
Train on 80 samples, validate on 10 samples
Epoch 1/500
80/80 [==============================] - 5s 62ms/step - loss: 0.9780 - accuracy:
0.4750 - val_loss: 0.6387 - val_accuracy: 0.7000
Epoch 2/500
80/80 [==============================] - 0s 256us/step - loss: 0.9506 -
accuracy: 0.4750 - val_loss: 0.6279 - val_accuracy: 0.7000
Epoch 3/500
80/80 [==============================] - 0s 477us/step - loss: 0.9321 -
accuracy: 0.4750 - val_loss: 0.6203 - val_accuracy: 0.7000
Epoch 4/500
80/80 [==============================] - 0s 289us/step - loss: 0.9082 -
accuracy: 0.4750 - val_loss: 0.6179 - val_accuracy: 0.7000
Epoch 5/500
80/80 [==============================] - 0s 244us/step - loss: 0.8905 -
accuracy: 0.4750 - val_loss: 0.6102 - val_accuracy: 0.7000
Epoch 6/500
80/80 [==============================] - 0s 341us/step - loss: 0.8704 -
```

```
accuracy: 0.4750 - val_loss: 0.6024 - val_accuracy: 0.7000
Epoch 7/500
80/80 [==============================] - 0s 263us/step - loss: 0.8522 -
accuracy: 0.4750 - val_loss: 0.6023 - val_accuracy: 0.7000
Epoch 8/500
80/80 [==============================] - 0s 278us/step - loss: 0.8354 -
accuracy: 0.4750 - val_loss: 0.6035 - val_accuracy: 0.7000
Epoch 9/500
80/80 [==============================] - 0s 221us/step - loss: 0.8220 -
accuracy: 0.4750 - val_loss: 0.6053 - val_accuracy: 0.7000
Epoch 10/500
80/80 [==============================] - 0s 237us/step - loss: 0.8103 -
accuracy: 0.4750 - val_loss: 0.6051 - val_accuracy: 0.7000
Epoch 11/500
80/80 [==============================] - 0s 291us/step - loss: 0.8005 -
accuracy: 0.4750 - val_loss: 0.6047 - val_accuracy: 0.7000
Epoch 12/500
80/80 [==============================] - 0s 357us/step - loss: 0.7938 -
accuracy: 0.4750 - val_loss: 0.6045 - val_accuracy: 0.7000
Epoch 13/500
80/80 [==============================] - 0s 203us/step - loss: 0.7865 -
accuracy: 0.4750 - val_loss: 0.6045 - val_accuracy: 0.7000
Epoch 14/500
80/80 [==============================] - 0s 289us/step - loss: 0.7795 -
accuracy: 0.4750 - val_loss: 0.6047 - val_accuracy: 0.7000
Epoch 15/500
80/80 [==============================] - 0s 247us/step - loss: 0.7734 -
accuracy: 0.4750 - val_loss: 0.6050 - val_accuracy: 0.7000
Epoch 16/500
80/80 [==============================] - 0s 304us/step - loss: 0.7678 -
accuracy: 0.4750 - val_loss: 0.6055 - val_accuracy: 0.7000
Epoch 17/500
80/80 [==============================] - 0s 231us/step - loss: 0.7624 -
accuracy: 0.4750 - val_loss: 0.6061 - val_accuracy: 0.7000
Epoch 18/500
80/80 [==============================] - 0s 339us/step - loss: 0.7565 -
accuracy: 0.4750 - val_loss: 0.6069 - val_accuracy: 0.7000
Epoch 19/500
80/80 [==============================] - 0s 278us/step - loss: 0.7513 -
accuracy: 0.4750 - val_loss: 0.6077 - val_accuracy: 0.7000
Epoch 20/500
80/80 [==============================] - 0s 490us/step - loss: 0.7468 -
accuracy: 0.4750 - val_loss: 0.6087 - val_accuracy: 0.7000
Epoch 21/500
80/80 [==============================] - 0s 406us/step - loss: 0.7417 -
accuracy: 0.4750 - val_loss: 0.6097 - val_accuracy: 0.7000
Epoch 22/500
80/80 [==============================] - 0s 270us/step - loss: 0.7378 -
```

```
accuracy: 0.4750 - val_loss: 0.6109 - val_accuracy: 0.7000
Epoch 23/500
80/80 [==============================] - 0s 290us/step - loss: 0.7337 -
accuracy: 0.4750 - val_loss: 0.6120 - val_accuracy: 0.7000
Epoch 24/500
80/80 [==============================] - 0s 277us/step - loss: 0.7298 -
accuracy: 0.4750 - val_loss: 0.6132 - val_accuracy: 0.7000
Epoch 25/500
80/80 [==============================] - 0s 422us/step - loss: 0.7260 -
accuracy: 0.4750 - val_loss: 0.6145 - val_accuracy: 0.7000
Epoch 26/500
80/80 [==============================] - 0s 373us/step - loss: 0.7227 -
accuracy: 0.4750 - val_loss: 0.6158 - val_accuracy: 0.7000
Epoch 27/500
80/80 [==============================] - 0s 342us/step - loss: 0.7196 -
accuracy: 0.4750 - val_loss: 0.6172 - val_accuracy: 0.7000
Epoch 28/500
80/80 [==============================] - 0s 692us/step - loss: 0.7165 -
accuracy: 0.4750 - val_loss: 0.6186 - val_accuracy: 0.7000
Epoch 29/500
80/80 [==============================] - 0s 436us/step - loss: 0.7135 -
accuracy: 0.4750 - val_loss: 0.6200 - val_accuracy: 0.7000
Epoch 30/500
80/80 [==============================] - 0s 325us/step - loss: 0.7112 -
accuracy: 0.4750 - val_loss: 0.6215 - val_accuracy: 0.7000
Epoch 31/500
80/80 [==============================] - 0s 374us/step - loss: 0.7083 -
accuracy: 0.4750 - val_loss: 0.6229 - val_accuracy: 0.7000
Epoch 32/500
80/80 [==============================] - 0s 373us/step - loss: 0.7059 -
accuracy: 0.4750 - val_loss: 0.6243 - val_accuracy: 0.7000
Epoch 33/500
80/80 [==============================] - 0s 470us/step - loss: 0.7035 -
accuracy: 0.4750 - val_loss: 0.6257 - val_accuracy: 0.7000
Epoch 34/500
80/80 [==============================] - 0s 376us/step - loss: 0.7014 -
accuracy: 0.4750 - val_loss: 0.6272 - val_accuracy: 0.7000
Epoch 35/500
80/80 [==============================] - 0s 297us/step - loss: 0.6992 -
accuracy: 0.4750 - val_loss: 0.6286 - val_accuracy: 0.7000
Epoch 36/500
80/80 [==============================] - 0s 324us/step - loss: 0.6972 -
accuracy: 0.4750 - val_loss: 0.6302 - val_accuracy: 0.7000
Epoch 37/500
80/80 [==============================] - 0s 256us/step - loss: 0.6955 -
accuracy: 0.4750 - val_loss: 0.6318 - val_accuracy: 0.7000
Epoch 38/500
80/80 [==============================] - 0s 377us/step - loss: 0.6937 -
```

```
accuracy: 0.4750 - val_loss: 0.6334 - val_accuracy: 0.7000
Epoch 39/500
80/80 [==============================] - 0s 239us/step - loss: 0.6917 -
accuracy: 0.4750 - val_loss: 0.6349 - val_accuracy: 0.7000
Epoch 40/500
80/80 [==============================] - 0s 288us/step - loss: 0.6905 -
accuracy: 0.4750 - val_loss: 0.6363 - val_accuracy: 0.7000
Epoch 41/500
80/80 [==============================] - 0s 511us/step - loss: 0.6886 -
accuracy: 0.4750 - val_loss: 0.6375 - val_accuracy: 0.7000
Epoch 42/500
80/80 [==============================] - 0s 436us/step - loss: 0.6874 -
accuracy: 0.4750 - val_loss: 0.6388 - val_accuracy: 0.7000
Epoch 43/500
80/80 [==============================] - 0s 421us/step - loss: 0.6859 -
accuracy: 0.4750 - val_loss: 0.6400 - val_accuracy: 0.7000
Epoch 44/500
80/80 [==============================] - 0s 295us/step - loss: 0.6850 -
accuracy: 0.4750 - val_loss: 0.6413 - val_accuracy: 0.7000
Epoch 45/500
80/80 [==============================] - 0s 338us/step - loss: 0.6836 -
accuracy: 0.4750 - val_loss: 0.6424 - val_accuracy: 0.7000
Epoch 46/500
80/80 [==============================] - 0s 587us/step - loss: 0.6825 -
accuracy: 0.4750 - val_loss: 0.6435 - val_accuracy: 0.7000
Epoch 47/500
80/80 [==============================] - 0s 531us/step - loss: 0.6811 -
accuracy: 0.4750 - val_loss: 0.6445 - val_accuracy: 0.7000
Epoch 48/500
80/80 [==============================] - 0s 372us/step - loss: 0.6802 -
accuracy: 0.4750 - val_loss: 0.6457 - val_accuracy: 0.7000
Epoch 49/500
80/80 [==============================] - 0s 396us/step - loss: 0.6790 -
accuracy: 0.4750 - val_loss: 0.6468 - val_accuracy: 0.7000
Epoch 50/500
80/80 [==============================] - 0s 291us/step - loss: 0.6781 -
accuracy: 0.4750 - val_loss: 0.6480 - val_accuracy: 0.7000
Epoch 51/500
80/80 [==============================] - 0s 304us/step - loss: 0.6769 -
accuracy: 0.4750 - val_loss: 0.6490 - val_accuracy: 0.7000
Epoch 52/500
80/80 [==============================] - 0s 242us/step - loss: 0.6758 -
accuracy: 0.4750 - val_loss: 0.6498 - val_accuracy: 0.7000
Epoch 53/500
80/80 [==============================] - 0s 447us/step - loss: 0.6751 -
accuracy: 0.4750 - val_loss: 0.6507 - val_accuracy: 0.7000
Epoch 54/500
80/80 [==============================] - 0s 441us/step - loss: 0.6742 -
```

```
accuracy: 0.4750 - val_loss: 0.6516 - val_accuracy: 0.7000
Epoch 55/500
80/80 [==============================] - 0s 532us/step - loss: 0.6732 -
accuracy: 0.4750 - val_loss: 0.6522 - val_accuracy: 0.7000
Epoch 56/500
80/80 [==============================] - 0s 447us/step - loss: 0.6723 -
accuracy: 0.4750 - val_loss: 0.6526 - val_accuracy: 0.7000
Epoch 57/500
80/80 [==============================] - 0s 253us/step - loss: 0.6716 -
accuracy: 0.4750 - val_loss: 0.6530 - val_accuracy: 0.7000
Epoch 58/500
80/80 [==============================] - 0s 331us/step - loss: 0.6707 -
accuracy: 0.4750 - val_loss: 0.6526 - val_accuracy: 0.7000
Epoch 59/500
80/80 [==============================] - 0s 272us/step - loss: 0.6700 -
accuracy: 0.4750 - val_loss: 0.6529 - val_accuracy: 0.7000
Epoch 60/500
80/80 [==============================] - 0s 334us/step - loss: 0.6692 -
accuracy: 0.4750 - val_loss: 0.6533 - val_accuracy: 0.7000
Epoch 61/500
80/80 [==============================] - 0s 301us/step - loss: 0.6685 -
accuracy: 0.4750 - val_loss: 0.6531 - val_accuracy: 0.7000
Epoch 62/500
80/80 [==============================] - 0s 250us/step - loss: 0.6675 -
accuracy: 0.4750 - val_loss: 0.6526 - val_accuracy: 0.7000
Epoch 63/500
80/80 [==============================] - 0s 305us/step - loss: 0.6668 -
accuracy: 0.4750 - val_loss: 0.6515 - val_accuracy: 0.7000
Epoch 64/500
80/80 [==============================] - 0s 450us/step - loss: 0.6657 -
accuracy: 0.4750 - val_loss: 0.6527 - val_accuracy: 0.7000
Epoch 65/500
80/80 [==============================] - 0s 331us/step - loss: 0.6646 -
accuracy: 0.4750 - val_loss: 0.6529 - val_accuracy: 0.7000
Epoch 66/500
80/80 [==============================] - 0s 297us/step - loss: 0.6631 -
accuracy: 0.4750 - val_loss: 0.6539 - val_accuracy: 0.7000
Epoch 67/500
80/80 [==============================] - 0s 289us/step - loss: 0.6618 -
accuracy: 0.4750 - val_loss: 0.6547 - val_accuracy: 0.8000
Epoch 68/500
80/80 [==============================] - 0s 398us/step - loss: 0.6605 -
accuracy: 0.4875 - val_loss: 0.6577 - val_accuracy: 0.9000
Epoch 69/500
80/80 [==============================] - 0s 259us/step - loss: 0.6585 -
accuracy: 0.6250 - val_loss: 0.6573 - val_accuracy: 0.8000
Epoch 70/500
80/80 [==============================] - 0s 230us/step - loss: 0.6568 -
```

```
accuracy: 0.5750 - val_loss: 0.6555 - val_accuracy: 0.7000
Epoch 71/500
80/80 [==============================] - 0s 242us/step - loss: 0.6549 -
accuracy: 0.5750 - val_loss: 0.6577 - val_accuracy: 0.6000
Epoch 72/500
80/80 [==============================] - 0s 377us/step - loss: 0.6532 -
accuracy: 0.6375 - val_loss: 0.6615 - val_accuracy: 0.5000
Epoch 73/500
80/80 [==============================] - 0s 472us/step - loss: 0.6517 -
accuracy: 0.7125 - val_loss: 0.6626 - val_accuracy: 0.5000
Epoch 74/500
80/80 [==============================] - 0s 257us/step - loss: 0.6501 -
accuracy: 0.6875 - val_loss: 0.6612 - val_accuracy: 0.5000
Epoch 75/500
80/80 [==============================] - 0s 326us/step - loss: 0.6485 -
accuracy: 0.7000 - val_loss: 0.6614 - val_accuracy: 0.5000
Epoch 76/500
80/80 [==============================] - 0s 349us/step - loss: 0.6470 -
accuracy: 0.7000 - val_loss: 0.6632 - val_accuracy: 0.5000
Epoch 77/500
80/80 [==============================] - 0s 261us/step - loss: 0.6454 -
accuracy: 0.6875 - val_loss: 0.6661 - val_accuracy: 0.5000
Epoch 78/500
80/80 [==============================] - 0s 311us/step - loss: 0.6441 -
accuracy: 0.6500 - val_loss: 0.6681 - val_accuracy: 0.5000
Epoch 79/500
80/80 [==============================] - 0s 343us/step - loss: 0.6427 -
accuracy: 0.6500 - val_loss: 0.6684 - val_accuracy: 0.5000
Epoch 80/500
80/80 [==============================] - 0s 302us/step - loss: 0.6413 -
accuracy: 0.6625 - val_loss: 0.6689 - val_accuracy: 0.5000
Epoch 81/500
80/80 [==============================] - 0s 297us/step - loss: 0.6401 -
accuracy: 0.6500 - val_loss: 0.6690 - val_accuracy: 0.5000
Epoch 82/500
80/80 [==============================] - 0s 534us/step - loss: 0.6390 -
accuracy: 0.6500 - val_loss: 0.6682 - val_accuracy: 0.5000
Epoch 83/500
80/80 [==============================] - 0s 283us/step - loss: 0.6382 -
accuracy: 0.6375 - val_loss: 0.6683 - val_accuracy: 0.5000
Epoch 84/500
80/80 [==============================] - 0s 226us/step - loss: 0.6369 -
accuracy: 0.6625 - val_loss: 0.6645 - val_accuracy: 0.5000
Epoch 85/500
80/80 [==============================] - 0s 285us/step - loss: 0.6359 -
accuracy: 0.6750 - val_loss: 0.6632 - val_accuracy: 0.5000
Epoch 86/500
80/80 [==============================] - 0s 412us/step - loss: 0.6354 -
```

```
accuracy: 0.6500 - val_loss: 0.6670 - val_accuracy: 0.5000
Epoch 87/500
80/80 [==============================] - 0s 272us/step - loss: 0.6339 -
accuracy: 0.6625 - val_loss: 0.6661 - val_accuracy: 0.5000
Epoch 88/500
80/80 [==============================] - 0s 473us/step - loss: 0.6331 -
accuracy: 0.6500 - val_loss: 0.6663 - val_accuracy: 0.5000
Epoch 89/500
80/80 [==============================] - 0s 497us/step - loss: 0.6318 -
accuracy: 0.6750 - val_loss: 0.6610 - val_accuracy: 0.5000
Epoch 90/500
80/80 [==============================] - 0s 304us/step - loss: 0.6305 -
accuracy: 0.6875 - val_loss: 0.6587 - val_accuracy: 0.5000
Epoch 91/500
80/80 [==============================] - 0s 376us/step - loss: 0.6306 -
accuracy: 0.6750 - val_loss: 0.6585 - val_accuracy: 0.5000
Epoch 92/500
80/80 [==============================] - 0s 281us/step - loss: 0.6286 -
accuracy: 0.6875 - val_loss: 0.6630 - val_accuracy: 0.5000
Epoch 93/500
80/80 [==============================] - 0s 291us/step - loss: 0.6286 -
accuracy: 0.6375 - val_loss: 0.6662 - val_accuracy: 0.4000
Epoch 94/500
80/80 [==============================] - 0s 294us/step - loss: 0.6282 -
accuracy: 0.6500 - val_loss: 0.6627 - val_accuracy: 0.5000
Epoch 95/500
80/80 [==============================] - 0s 637us/step - loss: 0.6264 -
accuracy: 0.6625 - val_loss: 0.6591 - val_accuracy: 0.5000
Epoch 96/500
80/80 [==============================] - 0s 511us/step - loss: 0.6256 -
accuracy: 0.6875 - val_loss: 0.6561 - val_accuracy: 0.5000
Epoch 97/500
80/80 [==============================] - 0s 255us/step - loss: 0.6260 -
accuracy: 0.6750 - val_loss: 0.6555 - val_accuracy: 0.5000
Epoch 98/500
80/80 [==============================] - 0s 336us/step - loss: 0.6241 -
accuracy: 0.6750 - val_loss: 0.6613 - val_accuracy: 0.5000
Epoch 99/500
80/80 [==============================] - 0s 256us/step - loss: 0.6247 -
accuracy: 0.6625 - val_loss: 0.6642 - val_accuracy: 0.5000
Epoch 100/500
80/80 [==============================] - 0s 294us/step - loss: 0.6242 -
accuracy: 0.6500 - val_loss: 0.6614 - val_accuracy: 0.5000
Epoch 101/500
80/80 [==============================] - 0s 290us/step - loss: 0.6227 -
accuracy: 0.6625 - val_loss: 0.6564 - val_accuracy: 0.5000
Epoch 102/500
80/80 [==============================] - 0s 298us/step - loss: 0.6221 -
```

```
accuracy: 0.6875 - val_loss: 0.6531 - val_accuracy: 0.5000
Epoch 103/500
80/80 [==============================] - 0s 327us/step - loss: 0.6228 -
accuracy: 0.6750 - val_loss: 0.6564 - val_accuracy: 0.5000
Epoch 104/500
80/80 [==============================] - 0s 306us/step - loss: 0.6215 -
accuracy: 0.6750 - val_loss: 0.6552 - val_accuracy: 0.5000
Epoch 105/500
80/80 [==============================] - 0s 477us/step - loss: 0.6220 -
accuracy: 0.6625 - val_loss: 0.6604 - val_accuracy: 0.5000
Epoch 106/500
80/80 [==============================] - 0s 225us/step - loss: 0.6209 -
accuracy: 0.6625 - val_loss: 0.6589 - val_accuracy: 0.5000
Epoch 107/500
80/80 [==============================] - 0s 345us/step - loss: 0.6200 -
accuracy: 0.6625 - val_loss: 0.6564 - val_accuracy: 0.5000
Epoch 108/500
80/80 [==============================] - 0s 318us/step - loss: 0.6186 -
accuracy: 0.6750 - val_loss: 0.6523 - val_accuracy: 0.5000
Epoch 109/500
80/80 [==============================] - 0s 339us/step - loss: 0.6194 -
accuracy: 0.6750 - val_loss: 0.6489 - val_accuracy: 0.5000
Epoch 110/500
80/80 [==============================] - 0s 286us/step - loss: 0.6191 -
accuracy: 0.6750 - val_loss: 0.6508 - val_accuracy: 0.5000
Epoch 111/500
80/80 [==============================] - 0s 289us/step - loss: 0.6174 -
accuracy: 0.6875 - val_loss: 0.6550 - val_accuracy: 0.5000
Epoch 112/500
80/80 [==============================] - 0s 978us/step - loss: 0.6176 -
accuracy: 0.6750 - val_loss: 0.6591 - val_accuracy: 0.5000
Epoch 113/500
80/80 [==============================] - 0s 448us/step - loss: 0.6179 -
accuracy: 0.6625 - val_loss: 0.6590 - val_accuracy: 0.5000
Epoch 114/500
80/80 [==============================] - 0s 503us/step - loss: 0.6174 -
accuracy: 0.6625 - val_loss: 0.6567 - val_accuracy: 0.5000
Epoch 115/500
80/80 [==============================] - 0s 431us/step - loss: 0.6161 -
accuracy: 0.6875 - val_loss: 0.6515 - val_accuracy: 0.5000
Epoch 116/500
80/80 [==============================] - 0s 348us/step - loss: 0.6155 -
accuracy: 0.6750 - val_loss: 0.6462 - val_accuracy: 0.5000
Epoch 117/500
80/80 [==============================] - 0s 394us/step - loss: 0.6170 -
accuracy: 0.6750 - val_loss: 0.6470 - val_accuracy: 0.5000
Epoch 118/500
80/80 [==============================] - 0s 378us/step - loss: 0.6153 -
```

```
accuracy: 0.6750 - val_loss: 0.6475 - val_accuracy: 0.5000
Epoch 119/500
80/80 [==============================] - 0s 436us/step - loss: 0.6146 -
accuracy: 0.6875 - val_loss: 0.6477 - val_accuracy: 0.5000
Epoch 120/500
80/80 [==============================] - 0s 373us/step - loss: 0.6142 -
accuracy: 0.6875 - val_loss: 0.6469 - val_accuracy: 0.5000
Epoch 121/500
80/80 [==============================] - 0s 307us/step - loss: 0.6142 -
accuracy: 0.6875 - val_loss: 0.6423 - val_accuracy: 0.5000
Epoch 122/500
80/80 [==============================] - 0s 441us/step - loss: 0.6138 -
accuracy: 0.6875 - val_loss: 0.6454 - val_accuracy: 0.5000
Epoch 123/500
80/80 [==============================] - 0s 241us/step - loss: 0.6131 -
accuracy: 0.6875 - val_loss: 0.6456 - val_accuracy: 0.5000
Epoch 124/500
80/80 [==============================] - 0s 300us/step - loss: 0.6127 -
accuracy: 0.6875 - val_loss: 0.6444 - val_accuracy: 0.5000
Epoch 125/500
80/80 [==============================] - 0s 792us/step - loss: 0.6125 -
accuracy: 0.6875 - val_loss: 0.6450 - val_accuracy: 0.5000
Epoch 126/500
80/80 [==============================] - 0s 431us/step - loss: 0.6121 -
accuracy: 0.6875 - val_loss: 0.6439 - val_accuracy: 0.5000
Epoch 127/500
80/80 [==============================] - 0s 587us/step - loss: 0.6121 -
accuracy: 0.6750 - val_loss: 0.6435 - val_accuracy: 0.5000
Epoch 128/500
80/80 [==============================] - 0s 527us/step - loss: 0.6120 -
accuracy: 0.6750 - val_loss: 0.6459 - val_accuracy: 0.5000
Epoch 129/500
80/80 [==============================] - 0s 607us/step - loss: 0.6111 -
accuracy: 0.6875 - val_loss: 0.6441 - val_accuracy: 0.5000
Epoch 130/500
80/80 [==============================] - 0s 249us/step - loss: 0.6108 -
accuracy: 0.6875 - val_loss: 0.6421 - val_accuracy: 0.5000
Epoch 131/500
80/80 [==============================] - 0s 268us/step - loss: 0.6106 -
accuracy: 0.6750 - val_loss: 0.6377 - val_accuracy: 0.5000
Epoch 132/500
80/80 [==============================] - 0s 611us/step - loss: 0.6105 -
accuracy: 0.6750 - val_loss: 0.6408 - val_accuracy: 0.5000
Epoch 133/500
80/80 [==============================] - 0s 623us/step - loss: 0.6101 -
accuracy: 0.6750 - val_loss: 0.6409 - val_accuracy: 0.5000
Epoch 134/500
80/80 [==============================] - 0s 1ms/step - loss: 0.6093 - accuracy:
```

```
0.6750 - val_loss: 0.6430 - val_accuracy: 0.5000
Epoch 135/500
80/80 [==============================] - 0s 405us/step - loss: 0.6092 -
accuracy: 0.6875 - val_loss: 0.6450 - val_accuracy: 0.5000
Epoch 136/500
80/80 [==============================] - 0s 337us/step - loss: 0.6088 -
accuracy: 0.6875 - val_loss: 0.6438 - val_accuracy: 0.5000
Epoch 137/500
80/80 [==============================] - 0s 362us/step - loss: 0.6081 -
accuracy: 0.6750 - val_loss: 0.6396 - val_accuracy: 0.5000
Epoch 138/500
80/80 [==============================] - 0s 521us/step - loss: 0.6095 -
accuracy: 0.6750 - val_loss: 0.6371 - val_accuracy: 0.5000
Epoch 139/500
80/80 [==============================] - 0s 378us/step - loss: 0.6085 -
accuracy: 0.6750 - val_loss: 0.6438 - val_accuracy: 0.5000
Epoch 140/500
80/80 [==============================] - 0s 354us/step - loss: 0.6075 -
accuracy: 0.6875 - val_loss: 0.6448 - val_accuracy: 0.5000
Epoch 141/500
80/80 [==============================] - 0s 270us/step - loss: 0.6078 -
accuracy: 0.6750 - val_loss: 0.6471 - val_accuracy: 0.5000
Epoch 142/500
80/80 [==============================] - 0s 378us/step - loss: 0.6081 -
accuracy: 0.6750 - val_loss: 0.6472 - val_accuracy: 0.5000
Epoch 143/500
80/80 [==============================] - 0s 490us/step - loss: 0.6071 -
accuracy: 0.6750 - val_loss: 0.6477 - val_accuracy: 0.5000
Epoch 144/500
80/80 [==============================] - 0s 342us/step - loss: 0.6067 -
accuracy: 0.6750 - val_loss: 0.6435 - val_accuracy: 0.5000
Epoch 145/500
80/80 [==============================] - 0s 476us/step - loss: 0.6065 -
accuracy: 0.6750 - val_loss: 0.6423 - val_accuracy: 0.5000
Epoch 146/500
80/80 [==============================] - 0s 423us/step - loss: 0.6056 -
accuracy: 0.6750 - val_loss: 0.6371 - val_accuracy: 0.5000
Epoch 147/500
80/80 [==============================] - 0s 333us/step - loss: 0.6060 -
accuracy: 0.6750 - val_loss: 0.6371 - val_accuracy: 0.5000
Epoch 148/500
80/80 [==============================] - 0s 345us/step - loss: 0.6052 -
accuracy: 0.6750 - val_loss: 0.6368 - val_accuracy: 0.5000
Epoch 149/500
80/80 [==============================] - 0s 358us/step - loss: 0.6053 -
accuracy: 0.6750 - val_loss: 0.6402 - val_accuracy: 0.5000
Epoch 150/500
80/80 [==============================] - 0s 388us/step - loss: 0.6045 -
```

```
accuracy: 0.6875 - val_loss: 0.6420 - val_accuracy: 0.5000
Epoch 151/500
80/80 [==============================] - 0s 323us/step - loss: 0.6042 -
accuracy: 0.6875 - val_loss: 0.6412 - val_accuracy: 0.5000
Epoch 152/500
80/80 [==============================] - 0s 414us/step - loss: 0.6038 -
accuracy: 0.6875 - val_loss: 0.6404 - val_accuracy: 0.5000
Epoch 153/500
80/80 [==============================] - 0s 382us/step - loss: 0.6036 -
accuracy: 0.6875 - val_loss: 0.6403 - val_accuracy: 0.5000
Epoch 154/500
80/80 [==============================] - 0s 290us/step - loss: 0.6045 -
accuracy: 0.6750 - val_loss: 0.6426 - val_accuracy: 0.5000
Epoch 155/500
80/80 [==============================] - 0s 378us/step - loss: 0.6034 -
accuracy: 0.6750 - val_loss: 0.6439 - val_accuracy: 0.5000
Epoch 156/500
80/80 [==============================] - 0s 397us/step - loss: 0.6035 -
accuracy: 0.6750 - val_loss: 0.6387 - val_accuracy: 0.5000
Epoch 157/500
80/80 [==============================] - 0s 358us/step - loss: 0.6033 -
accuracy: 0.6750 - val_loss: 0.6400 - val_accuracy: 0.5000
Epoch 158/500
80/80 [==============================] - 0s 688us/step - loss: 0.6023 -
accuracy: 0.6750 - val_loss: 0.6411 - val_accuracy: 0.5000
Epoch 159/500
80/80 [==============================] - 0s 431us/step - loss: 0.6023 -
accuracy: 0.6875 - val_loss: 0.6399 - val_accuracy: 0.5000
Epoch 160/500
80/80 [==============================] - 0s 608us/step - loss: 0.6021 -
accuracy: 0.6750 - val_loss: 0.6403 - val_accuracy: 0.5000
Epoch 161/500
80/80 [==============================] - 0s 393us/step - loss: 0.6015 -
accuracy: 0.6875 - val_loss: 0.6404 - val_accuracy: 0.5000
Epoch 162/500
80/80 [==============================] - 0s 373us/step - loss: 0.6013 -
accuracy: 0.6875 - val_loss: 0.6398 - val_accuracy: 0.5000
Epoch 163/500
80/80 [==============================] - 0s 234us/step - loss: 0.6026 -
accuracy: 0.6750 - val_loss: 0.6332 - val_accuracy: 0.5000
Epoch 164/500
80/80 [==============================] - 0s 349us/step - loss: 0.6009 -
accuracy: 0.6875 - val_loss: 0.6401 - val_accuracy: 0.5000
Epoch 165/500
80/80 [==============================] - 0s 554us/step - loss: 0.6009 -
accuracy: 0.6750 - val_loss: 0.6478 - val_accuracy: 0.5000
Epoch 166/500
80/80 [==============================] - 0s 394us/step - loss: 0.6017 -
```

```
accuracy: 0.6875 - val_loss: 0.6482 - val_accuracy: 0.5000
Epoch 167/500
80/80 [==============================] - 0s 475us/step - loss: 0.6013 -
accuracy: 0.6750 - val_loss: 0.6414 - val_accuracy: 0.5000
Epoch 168/500
80/80 [==============================] - 0s 416us/step - loss: 0.6017 -
accuracy: 0.6875 - val_loss: 0.6346 - val_accuracy: 0.5000
Epoch 169/500
80/80 [==============================] - 0s 422us/step - loss: 0.6038 -
accuracy: 0.6750 - val_loss: 0.6416 - val_accuracy: 0.5000
Epoch 170/500
80/80 [==============================] - 0s 555us/step - loss: 0.6000 -
accuracy: 0.7000 - val_loss: 0.6542 - val_accuracy: 0.5000
Epoch 171/500
80/80 [==============================] - 0s 295us/step - loss: 0.6037 -
accuracy: 0.6500 - val_loss: 0.6566 - val_accuracy: 0.5000
Epoch 172/500
80/80 [==============================] - 0s 578us/step - loss: 0.6022 -
accuracy: 0.6500 - val_loss: 0.6486 - val_accuracy: 0.5000
Epoch 173/500
80/80 [==============================] - 0s 361us/step - loss: 0.6010 -
accuracy: 0.6750 - val_loss: 0.6329 - val_accuracy: 0.5000
Epoch 174/500
80/80 [==============================] - 0s 218us/step - loss: 0.6003 -
accuracy: 0.6750 - val_loss: 0.6301 - val_accuracy: 0.5000
Epoch 175/500
80/80 [==============================] - 0s 269us/step - loss: 0.5986 -
accuracy: 0.6750 - val_loss: 0.6413 - val_accuracy: 0.5000
Epoch 176/500
80/80 [==============================] - 0s 219us/step - loss: 0.5980 -
accuracy: 0.6750 - val_loss: 0.6447 - val_accuracy: 0.5000
Epoch 177/500
80/80 [==============================] - 0s 235us/step - loss: 0.5989 -
accuracy: 0.6750 - val_loss: 0.6378 - val_accuracy: 0.5000
Epoch 178/500
80/80 [==============================] - 0s 767us/step - loss: 0.5973 -
accuracy: 0.6875 - val_loss: 0.6377 - val_accuracy: 0.5000
Epoch 179/500
80/80 [==============================] - 0s 336us/step - loss: 0.5972 -
accuracy: 0.6875 - val_loss: 0.6378 - val_accuracy: 0.5000
Epoch 180/500
80/80 [==============================] - 0s 252us/step - loss: 0.5965 -
accuracy: 0.6875 - val_loss: 0.6384 - val_accuracy: 0.5000
Epoch 181/500
80/80 [==============================] - 0s 333us/step - loss: 0.5966 -
accuracy: 0.6750 - val_loss: 0.6370 - val_accuracy: 0.5000
Epoch 182/500
80/80 [==============================] - 0s 402us/step - loss: 0.5968 -
```

```
accuracy: 0.6750 - val_loss: 0.6361 - val_accuracy: 0.5000
Epoch 183/500
80/80 [==============================] - 0s 336us/step - loss: 0.5964 -
accuracy: 0.6750 - val_loss: 0.6337 - val_accuracy: 0.5000
Epoch 184/500
80/80 [==============================] - 0s 385us/step - loss: 0.5961 -
accuracy: 0.6875 - val_loss: 0.6339 - val_accuracy: 0.5000
Epoch 185/500
80/80 [==============================] - 0s 988us/step - loss: 0.5967 -
accuracy: 0.6750 - val_loss: 0.6408 - val_accuracy: 0.5000
Epoch 186/500
80/80 [==============================] - 0s 284us/step - loss: 0.5959 -
accuracy: 0.6750 - val_loss: 0.6416 - val_accuracy: 0.5000
Epoch 187/500
80/80 [==============================] - 0s 606us/step - loss: 0.5961 -
accuracy: 0.6875 - val_loss: 0.6351 - val_accuracy: 0.5000
Epoch 188/500
80/80 [==============================] - 0s 482us/step - loss: 0.5955 -
accuracy: 0.6875 - val_loss: 0.6347 - val_accuracy: 0.5000
Epoch 189/500
80/80 [==============================] - 0s 337us/step - loss: 0.5963 -
accuracy: 0.6750 - val_loss: 0.6347 - val_accuracy: 0.5000
Epoch 190/500
80/80 [==============================] - 0s 491us/step - loss: 0.5952 -
accuracy: 0.6750 - val_loss: 0.6286 - val_accuracy: 0.5000
Epoch 191/500
80/80 [==============================] - 0s 455us/step - loss: 0.5965 -
accuracy: 0.6750 - val_loss: 0.6314 - val_accuracy: 0.5000
Epoch 192/500
80/80 [==============================] - 0s 282us/step - loss: 0.5950 -
accuracy: 0.6875 - val_loss: 0.6303 - val_accuracy: 0.5000
Epoch 193/500
80/80 [==============================] - 0s 310us/step - loss: 0.5939 -
accuracy: 0.6750 - val_loss: 0.6352 - val_accuracy: 0.5000
Epoch 194/500
80/80 [==============================] - 0s 311us/step - loss: 0.5945 -
accuracy: 0.6750 - val_loss: 0.6333 - val_accuracy: 0.5000
Epoch 195/500
80/80 [==============================] - 0s 405us/step - loss: 0.5936 -
accuracy: 0.6750 - val_loss: 0.6270 - val_accuracy: 0.5000
Epoch 196/500
80/80 [==============================] - 0s 286us/step - loss: 0.5939 -
accuracy: 0.6750 - val_loss: 0.6297 - val_accuracy: 0.5000
Epoch 197/500
80/80 [==============================] - 0s 473us/step - loss: 0.5945 -
accuracy: 0.6875 - val_loss: 0.6382 - val_accuracy: 0.5000
Epoch 198/500
80/80 [==============================] - 0s 590us/step - loss: 0.5936 -
```

```
accuracy: 0.6750 - val_loss: 0.6362 - val_accuracy: 0.5000
Epoch 199/500
80/80 [==============================] - 0s 489us/step - loss: 0.5923 -
accuracy: 0.6875 - val_loss: 0.6293 - val_accuracy: 0.5000
Epoch 200/500
80/80 [==============================] - 0s 630us/step - loss: 0.5920 -
accuracy: 0.6750 - val_loss: 0.6234 - val_accuracy: 0.5000
Epoch 201/500
80/80 [==============================] - 0s 557us/step - loss: 0.5957 -
accuracy: 0.6625 - val_loss: 0.6224 - val_accuracy: 0.5000
Epoch 202/500
80/80 [==============================] - 0s 342us/step - loss: 0.5914 -
accuracy: 0.6750 - val_loss: 0.6425 - val_accuracy: 0.5000
Epoch 203/500
80/80 [==============================] - 0s 340us/step - loss: 0.5946 -
accuracy: 0.6750 - val_loss: 0.6535 - val_accuracy: 0.5000
Epoch 204/500
80/80 [==============================] - 0s 339us/step - loss: 0.5956 -
accuracy: 0.6500 - val_loss: 0.6461 - val_accuracy: 0.5000
Epoch 205/500
80/80 [==============================] - 0s 456us/step - loss: 0.5953 -
accuracy: 0.6500 - val_loss: 0.6403 - val_accuracy: 0.5000
Epoch 206/500
80/80 [==============================] - 0s 638us/step - loss: 0.5936 -
accuracy: 0.6750 - val_loss: 0.6385 - val_accuracy: 0.5000
Epoch 207/500
80/80 [==============================] - 0s 322us/step - loss: 0.5916 -
accuracy: 0.6875 - val_loss: 0.6359 - val_accuracy: 0.5000
Epoch 208/500
80/80 [==============================] - 0s 543us/step - loss: 0.5907 -
accuracy: 0.6750 - val_loss: 0.6256 - val_accuracy: 0.5000
Epoch 209/500
80/80 [==============================] - 0s 291us/step - loss: 0.5933 -
accuracy: 0.6750 - val_loss: 0.6276 - val_accuracy: 0.5000
Epoch 210/500
80/80 [==============================] - 0s 283us/step - loss: 0.5897 -
accuracy: 0.6875 - val_loss: 0.6351 - val_accuracy: 0.5000
Epoch 211/500
80/80 [==============================] - 0s 378us/step - loss: 0.5897 -
accuracy: 0.6875 - val_loss: 0.6394 - val_accuracy: 0.5000
Epoch 212/500
80/80 [==============================] - 0s 591us/step - loss: 0.5924 -
accuracy: 0.6750 - val_loss: 0.6429 - val_accuracy: 0.5000
Epoch 213/500
80/80 [==============================] - 0s 663us/step - loss: 0.5913 -
accuracy: 0.6875 - val_loss: 0.6409 - val_accuracy: 0.5000
Epoch 214/500
80/80 [==============================] - 0s 356us/step - loss: 0.5900 -
```

```
accuracy: 0.6750 - val_loss: 0.6368 - val_accuracy: 0.5000
Epoch 215/500
80/80 [==============================] - 0s 462us/step - loss: 0.5931 -
accuracy: 0.6750 - val_loss: 0.6241 - val_accuracy: 0.5000
Epoch 216/500
80/80 [==============================] - 0s 447us/step - loss: 0.5894 -
accuracy: 0.6750 - val_loss: 0.6304 - val_accuracy: 0.5000
Epoch 217/500
80/80 [==============================] - 0s 332us/step - loss: 0.5892 -
accuracy: 0.6750 - val_loss: 0.6419 - val_accuracy: 0.5000
Epoch 218/500
80/80 [==============================] - 0s 397us/step - loss: 0.5912 -
accuracy: 0.6625 - val_loss: 0.6457 - val_accuracy: 0.5000
Epoch 219/500
80/80 [==============================] - 0s 431us/step - loss: 0.5912 -
accuracy: 0.6625 - val_loss: 0.6477 - val_accuracy: 0.5000
Epoch 220/500
80/80 [==============================] - 0s 436us/step - loss: 0.5923 -
accuracy: 0.6750 - val_loss: 0.6357 - val_accuracy: 0.5000
Epoch 221/500
80/80 [==============================] - 0s 388us/step - loss: 0.5885 -
accuracy: 0.6750 - val_loss: 0.6329 - val_accuracy: 0.5000
Epoch 222/500
80/80 [==============================] - 0s 662us/step - loss: 0.5877 -
accuracy: 0.6750 - val_loss: 0.6347 - val_accuracy: 0.5000
Epoch 223/500
80/80 [==============================] - 0s 878us/step - loss: 0.5878 -
accuracy: 0.6750 - val_loss: 0.6330 - val_accuracy: 0.5000
Epoch 224/500
80/80 [==============================] - 0s 2ms/step - loss: 0.5872 - accuracy:
0.6750 - val_loss: 0.6297 - val_accuracy: 0.5000
Epoch 225/500
80/80 [==============================] - 0s 716us/step - loss: 0.5871 -
accuracy: 0.6750 - val_loss: 0.6305 - val_accuracy: 0.5000
Epoch 226/500
80/80 [==============================] - 0s 681us/step - loss: 0.5891 -
accuracy: 0.6750 - val_loss: 0.6324 - val_accuracy: 0.5000
Epoch 227/500
80/80 [==============================] - 0s 426us/step - loss: 0.5868 -
accuracy: 0.6875 - val_loss: 0.6335 - val_accuracy: 0.5000
Epoch 228/500
80/80 [==============================] - 0s 2ms/step - loss: 0.5886 - accuracy:
0.6750 - val_loss: 0.6402 - val_accuracy: 0.5000
Epoch 229/500
80/80 [==============================] - 0s 350us/step - loss: 0.5883 -
accuracy: 0.6750 - val_loss: 0.6385 - val_accuracy: 0.5000
Epoch 230/500
80/80 [==============================] - 0s 371us/step - loss: 0.5865 -
```

```
accuracy: 0.6875 - val_loss: 0.6302 - val_accuracy: 0.5000
Epoch 231/500
80/80 [==============================] - 0s 496us/step - loss: 0.5861 -
accuracy: 0.6875 - val_loss: 0.6275 - val_accuracy: 0.5000
Epoch 232/500
80/80 [==============================] - 0s 400us/step - loss: 0.5864 -
accuracy: 0.6875 - val_loss: 0.6310 - val_accuracy: 0.5000
Epoch 233/500
80/80 [==============================] - 0s 704us/step - loss: 0.5883 -
accuracy: 0.6750 - val_loss: 0.6322 - val_accuracy: 0.5000
Epoch 234/500
80/80 [==============================] - 0s 267us/step - loss: 0.5849 -
accuracy: 0.6750 - val_loss: 0.6291 - val_accuracy: 0.5000
Epoch 235/500
80/80 [==============================] - 0s 340us/step - loss: 0.5845 -
accuracy: 0.6750 - val_loss: 0.6272 - val_accuracy: 0.5000
Epoch 236/500
80/80 [==============================] - 0s 358us/step - loss: 0.5858 -
accuracy: 0.6750 - val_loss: 0.6222 - val_accuracy: 0.5000
Epoch 237/500
80/80 [==============================] - 0s 350us/step - loss: 0.5858 -
accuracy: 0.6750 - val_loss: 0.6273 - val_accuracy: 0.5000
Epoch 238/500
80/80 [==============================] - 0s 397us/step - loss: 0.5841 -
accuracy: 0.6875 - val_loss: 0.6254 - val_accuracy: 0.5000
Epoch 239/500
80/80 [==============================] - 0s 799us/step - loss: 0.5829 -
accuracy: 0.6875 - val_loss: 0.6130 - val_accuracy: 0.5000
Epoch 240/500
80/80 [==============================] - 0s 539us/step - loss: 0.5844 -
accuracy: 0.7500 - val_loss: 0.6124 - val_accuracy: 0.5000
Epoch 241/500
80/80 [==============================] - 0s 484us/step - loss: 0.5808 -
accuracy: 0.7125 - val_loss: 0.6345 - val_accuracy: 0.5000
Epoch 242/500
80/80 [==============================] - 0s 375us/step - loss: 0.5809 -
accuracy: 0.7000 - val_loss: 0.6327 - val_accuracy: 0.5000
Epoch 243/500
80/80 [==============================] - 0s 577us/step - loss: 0.5796 -
accuracy: 0.6875 - val_loss: 0.6094 - val_accuracy: 0.7000
Epoch 244/500
80/80 [==============================] - 0s 369us/step - loss: 0.5813 -
accuracy: 0.7000 - val_loss: 0.6003 - val_accuracy: 0.7000
Epoch 245/500
80/80 [==============================] - 0s 446us/step - loss: 0.5789 -
accuracy: 0.7625 - val_loss: 0.6293 - val_accuracy: 0.5000
Epoch 246/500
80/80 [==============================] - 0s 870us/step - loss: 0.5820 -
```

```
accuracy: 0.7000 - val_loss: 0.6215 - val_accuracy: 0.5000
Epoch 247/500
80/80 [==============================] - 0s 433us/step - loss: 0.5766 -
accuracy: 0.7500 - val_loss: 0.5995 - val_accuracy: 0.7000
Epoch 248/500
80/80 [==============================] - 0s 441us/step - loss: 0.5873 -
accuracy: 0.6750 - val_loss: 0.6275 - val_accuracy: 0.6000
Epoch 249/500
80/80 [==============================] - 0s 469us/step - loss: 0.5836 -
accuracy: 0.7000 - val_loss: 0.6190 - val_accuracy: 0.6000
Epoch 250/500
80/80 [==============================] - 0s 398us/step - loss: 0.5814 -
accuracy: 0.7375 - val_loss: 0.6405 - val_accuracy: 0.5000
Epoch 251/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5801 - accuracy:
0.6750 - val_loss: 0.6228 - val_accuracy: 0.6000
Epoch 252/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5750 - accuracy:
0.7000 - val_loss: 0.6094 - val_accuracy: 0.6000
Epoch 253/500
80/80 [==============================] - 0s 937us/step - loss: 0.5845 -
accuracy: 0.6875 - val_loss: 0.5896 - val_accuracy: 0.7000
Epoch 254/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5789 - accuracy:
0.7250 - val_loss: 0.6238 - val_accuracy: 0.6000
Epoch 255/500
80/80 [==============================] - 0s 511us/step - loss: 0.5788 -
accuracy: 0.7125 - val_loss: 0.6386 - val_accuracy: 0.5000
Epoch 256/500
80/80 [==============================] - 0s 388us/step - loss: 0.5774 -
accuracy: 0.7125 - val_loss: 0.6012 - val_accuracy: 0.7000
Epoch 257/500
80/80 [==============================] - 0s 558us/step - loss: 0.5776 -
accuracy: 0.7375 - val_loss: 0.5954 - val_accuracy: 0.7000
Epoch 258/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5757 - accuracy:
0.7500 - val_loss: 0.5935 - val_accuracy: 0.7000
Epoch 259/500
80/80 [==============================] - 0s 599us/step - loss: 0.5734 -
accuracy: 0.7375 - val_loss: 0.6064 - val_accuracy: 0.6000
Epoch 260/500
80/80 [==============================] - 0s 428us/step - loss: 0.5752 -
accuracy: 0.7250 - val_loss: 0.6151 - val_accuracy: 0.6000
Epoch 261/500
80/80 [==============================] - 0s 402us/step - loss: 0.5750 -
accuracy: 0.7125 - val_loss: 0.6059 - val_accuracy: 0.7000
Epoch 262/500
80/80 [==============================] - 0s 484us/step - loss: 0.5702 -
```

```
accuracy: 0.7375 - val_loss: 0.5835 - val_accuracy: 0.7000
Epoch 263/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5835 - accuracy:
0.6875 - val_loss: 0.5864 - val_accuracy: 0.7000
Epoch 264/500
80/80 [==============================] - 0s 608us/step - loss: 0.5732 -
accuracy: 0.7000 - val_loss: 0.6203 - val_accuracy: 0.5000
Epoch 265/500
80/80 [==============================] - 0s 344us/step - loss: 0.5725 -
accuracy: 0.7125 - val_loss: 0.6147 - val_accuracy: 0.6000
Epoch 266/500
80/80 [==============================] - 0s 471us/step - loss: 0.5695 -
accuracy: 0.7250 - val_loss: 0.5898 - val_accuracy: 0.7000
Epoch 267/500
80/80 [==============================] - 0s 442us/step - loss: 0.5794 -
accuracy: 0.6875 - val_loss: 0.6017 - val_accuracy: 0.8000
Epoch 268/500
80/80 [==============================] - 0s 506us/step - loss: 0.5712 -
accuracy: 0.7500 - val_loss: 0.6382 - val_accuracy: 0.5000
Epoch 269/500
80/80 [==============================] - 0s 471us/step - loss: 0.5766 -
accuracy: 0.7000 - val_loss: 0.6182 - val_accuracy: 0.5000
Epoch 270/500
80/80 [==============================] - 0s 301us/step - loss: 0.5723 -
accuracy: 0.7125 - val_loss: 0.5812 - val_accuracy: 0.7000
Epoch 271/500
80/80 [==============================] - 0s 736us/step - loss: 0.5725 -
accuracy: 0.7250 - val_loss: 0.5887 - val_accuracy: 0.7000
Epoch 272/500
80/80 [==============================] - 0s 526us/step - loss: 0.5808 -
accuracy: 0.6500 - val_loss: 0.6363 - val_accuracy: 0.5000
Epoch 273/500
80/80 [==============================] - 0s 690us/step - loss: 0.5716 -
accuracy: 0.7125 - val_loss: 0.6053 - val_accuracy: 0.6000
Epoch 274/500
80/80 [==============================] - 0s 432us/step - loss: 0.5638 -
accuracy: 0.7625 - val_loss: 0.5621 - val_accuracy: 0.7000
Epoch 275/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5742 - accuracy:
0.7125 - val_loss: 0.5786 - val_accuracy: 0.7000
Epoch 276/500
80/80 [==============================] - 0s 478us/step - loss: 0.5705 -
accuracy: 0.7250 - val_loss: 0.6268 - val_accuracy: 0.5000
Epoch 277/500
80/80 [==============================] - 0s 640us/step - loss: 0.5692 -
accuracy: 0.7250 - val_loss: 0.6145 - val_accuracy: 0.6000
Epoch 278/500
80/80 [==============================] - 0s 642us/step - loss: 0.5664 -
```

```
accuracy: 0.7250 - val_loss: 0.5935 - val_accuracy: 0.8000
Epoch 279/500
80/80 [==============================] - 0s 530us/step - loss: 0.5629 -
accuracy: 0.7750 - val_loss: 0.5757 - val_accuracy: 0.7000
Epoch 280/500
80/80 [==============================] - 0s 328us/step - loss: 0.5647 -
accuracy: 0.7250 - val_loss: 0.5770 - val_accuracy: 0.7000
Epoch 281/500
80/80 [==============================] - 0s 292us/step - loss: 0.5684 -
accuracy: 0.7250 - val_loss: 0.6140 - val_accuracy: 0.6000
Epoch 282/500
80/80 [==============================] - 0s 342us/step - loss: 0.5678 -
accuracy: 0.7125 - val_loss: 0.6636 - val_accuracy: 0.5000
Epoch 283/500
80/80 [==============================] - 0s 452us/step - loss: 0.5844 -
accuracy: 0.6875 - val_loss: 0.6634 - val_accuracy: 0.5000
Epoch 284/500
80/80 [==============================] - 0s 488us/step - loss: 0.5846 -
accuracy: 0.6875 - val_loss: 0.6703 - val_accuracy: 0.5000
Epoch 285/500
80/80 [==============================] - 0s 568us/step - loss: 0.5869 -
accuracy: 0.6750 - val_loss: 0.6777 - val_accuracy: 0.4000
Epoch 286/500
80/80 [==============================] - 0s 450us/step - loss: 0.5888 -
accuracy: 0.6625 - val_loss: 0.6774 - val_accuracy: 0.5000
Epoch 287/500
80/80 [==============================] - 0s 358us/step - loss: 0.5883 -
accuracy: 0.6625 - val_loss: 0.6732 - val_accuracy: 0.5000
Epoch 288/500
80/80 [==============================] - 0s 735us/step - loss: 0.5862 -
accuracy: 0.6875 - val_loss: 0.6670 - val_accuracy: 0.5000
Epoch 289/500
80/80 [==============================] - 0s 548us/step - loss: 0.5882 -
accuracy: 0.6875 - val_loss: 0.6628 - val_accuracy: 0.5000
Epoch 290/500
80/80 [==============================] - 0s 952us/step - loss: 0.5858 -
accuracy: 0.6875 - val_loss: 0.6696 - val_accuracy: 0.5000
Epoch 291/500
80/80 [==============================] - 0s 641us/step - loss: 0.5865 -
accuracy: 0.6875 - val_loss: 0.6714 - val_accuracy: 0.5000
Epoch 292/500
80/80 [==============================] - 0s 737us/step - loss: 0.5856 -
accuracy: 0.6875 - val_loss: 0.6676 - val_accuracy: 0.5000
Epoch 293/500
80/80 [==============================] - 0s 665us/step - loss: 0.5857 -
accuracy: 0.6750 - val_loss: 0.6609 - val_accuracy: 0.5000
Epoch 294/500
80/80 [==============================] - 0s 556us/step - loss: 0.5851 -
```

```
accuracy: 0.6875 - val_loss: 0.6634 - val_accuracy: 0.5000
Epoch 295/500
80/80 [==============================] - 0s 342us/step - loss: 0.5851 -
accuracy: 0.6875 - val_loss: 0.6678 - val_accuracy: 0.5000
Epoch 296/500
80/80 [==============================] - 0s 379us/step - loss: 0.5843 -
accuracy: 0.6750 - val_loss: 0.6676 - val_accuracy: 0.5000
Epoch 297/500
80/80 [==============================] - 0s 463us/step - loss: 0.5847 -
accuracy: 0.6750 - val_loss: 0.6674 - val_accuracy: 0.5000
Epoch 298/500
80/80 [==============================] - 0s 409us/step - loss: 0.5882 -
accuracy: 0.6750 - val_loss: 0.6636 - val_accuracy: 0.5000
Epoch 299/500
80/80 [==============================] - 0s 340us/step - loss: 0.5830 -
accuracy: 0.6875 - val_loss: 0.6671 - val_accuracy: 0.5000
Epoch 300/500
80/80 [==============================] - 0s 2ms/step - loss: 0.5831 - accuracy:
0.6750 - val_loss: 0.6675 - val_accuracy: 0.5000
Epoch 301/500
80/80 [==============================] - 0s 376us/step - loss: 0.5832 -
accuracy: 0.6750 - val_loss: 0.6678 - val_accuracy: 0.5000
Epoch 302/500
80/80 [==============================] - 0s 836us/step - loss: 0.5829 -
accuracy: 0.6750 - val_loss: 0.6663 - val_accuracy: 0.5000
Epoch 303/500
80/80 [==============================] - 0s 663us/step - loss: 0.5802 -
accuracy: 0.6750 - val_loss: 0.6630 - val_accuracy: 0.5000
Epoch 304/500
80/80 [==============================] - 0s 676us/step - loss: 0.5773 -
accuracy: 0.6750 - val_loss: 0.6476 - val_accuracy: 0.5000
Epoch 305/500
80/80 [==============================] - 0s 2ms/step - loss: 0.5740 - accuracy:
0.7125 - val_loss: 0.5932 - val_accuracy: 0.6000
Epoch 306/500
80/80 [==============================] - 0s 329us/step - loss: 0.5646 -
accuracy: 0.7125 - val_loss: 0.5748 - val_accuracy: 0.7000
Epoch 307/500
80/80 [==============================] - 0s 310us/step - loss: 0.5671 -
accuracy: 0.7125 - val_loss: 0.5968 - val_accuracy: 0.8000
Epoch 308/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5576 - accuracy:
0.7000 - val_loss: 0.6420 - val_accuracy: 0.5000
Epoch 309/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5746 - accuracy:
0.6750 - val_loss: 0.6616 - val_accuracy: 0.5000
Epoch 310/500
80/80 [==============================] - 0s 450us/step - loss: 0.5752 -
```

```
accuracy: 0.6875 - val_loss: 0.6473 - val_accuracy: 0.5000
Epoch 311/500
80/80 [==============================] - 0s 317us/step - loss: 0.5750 -
accuracy: 0.6875 - val_loss: 0.6199 - val_accuracy: 0.5000
Epoch 312/500
80/80 [==============================] - 0s 275us/step - loss: 0.5618 -
accuracy: 0.7500 - val_loss: 0.5804 - val_accuracy: 0.7000
Epoch 313/500
80/80 [==============================] - 0s 403us/step - loss: 0.5716 -
accuracy: 0.7250 - val_loss: 0.5880 - val_accuracy: 0.7000
Epoch 314/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5682 - accuracy:
0.7125 - val_loss: 0.6315 - val_accuracy: 0.5000
Epoch 315/500
80/80 [==============================] - 0s 655us/step - loss: 0.5653 -
accuracy: 0.6875 - val_loss: 0.6252 - val_accuracy: 0.5000
Epoch 316/500
80/80 [==============================] - 0s 587us/step - loss: 0.5655 -
accuracy: 0.7125 - val_loss: 0.5850 - val_accuracy: 0.7000
Epoch 317/500
80/80 [==============================] - 0s 454us/step - loss: 0.5638 -
accuracy: 0.7000 - val_loss: 0.5714 - val_accuracy: 0.7000
Epoch 318/500
80/80 [==============================] - 0s 6ms/step - loss: 0.5621 - accuracy:
0.7125 - val_loss: 0.5771 - val_accuracy: 0.8000
Epoch 319/500
80/80 [==============================] - 0s 653us/step - loss: 0.5538 -
accuracy: 0.8000 - val_loss: 0.6139 - val_accuracy: 0.6000
Epoch 320/500
80/80 [==============================] - 0s 410us/step - loss: 0.5588 -
accuracy: 0.7125 - val_loss: 0.6316 - val_accuracy: 0.5000
Epoch 321/500
32/80 [===========>…] - ETA: 0s - loss: 0.5326 - accuracy:
0.7188

/Users/javierserranomolina/miniconda3/envs/IA/lib/python3.7/site-
packages/keras/callbacks/callbacks.py:95: RuntimeWarning: Method
(on_train_batch_end) is slow compared to the batch update (0.103056). Check your
callbacks.
  % (hook_name, delta_t_median), RuntimeWarning)

80/80 [==============================] - 0s 702us/step - loss: 0.5639 -
accuracy: 0.7250 - val_loss: 0.6183 - val_accuracy: 0.5000
Epoch 322/500
80/80 [==============================] - 0s 2ms/step - loss: 0.5593 - accuracy:
0.7250 - val_loss: 0.5707 - val_accuracy: 0.8000
Epoch 323/500
80/80 [==============================] - 0s 667us/step - loss: 0.5560 -
accuracy: 0.7750 - val_loss: 0.5630 - val_accuracy: 0.7000
```

```
Epoch 324/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5612 - accuracy:
0.7250 - val_loss: 0.5930 - val_accuracy: 0.8000
Epoch 325/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5623 - accuracy:
0.7250 - val_loss: 0.6250 - val_accuracy: 0.6000
Epoch 326/500
80/80 [==============================] - 0s 473us/step - loss: 0.5607 -
accuracy: 0.7125 - val_loss: 0.6119 - val_accuracy: 0.6000
Epoch 327/500
80/80 [==============================] - 0s 487us/step - loss: 0.5542 -
accuracy: 0.7375 - val_loss: 0.5806 - val_accuracy: 0.8000
Epoch 328/500
80/80 [==============================] - 0s 479us/step - loss: 0.5540 -
accuracy: 0.7375 - val_loss: 0.5562 - val_accuracy: 0.7000
Epoch 329/500
80/80 [==============================] - 0s 2ms/step - loss: 0.5549 - accuracy:
0.7625 - val_loss: 0.5620 - val_accuracy: 0.7000
Epoch 330/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5510 - accuracy:
0.7625 - val_loss: 0.6117 - val_accuracy: 0.6000
Epoch 331/500

/Users/javierserranomolina/miniconda3/envs/IA/lib/python3.7/site-
packages/keras/callbacks/callbacks.py:95: RuntimeWarning: Method
(on_train_batch_end) is slow compared to the batch update (0.115933). Check your
callbacks.
  % (hook_name, delta_t_median), RuntimeWarning)

80/80 [==============================] - 0s 2ms/step - loss: 0.5601 - accuracy:
0.6875 - val_loss: 0.6178 - val_accuracy: 0.5000
Epoch 332/500
80/80 [==============================] - 0s 558us/step - loss: 0.5630 -
accuracy: 0.7250 - val_loss: 0.5744 - val_accuracy: 0.8000
Epoch 333/500
80/80 [==============================] - 0s 575us/step - loss: 0.5534 -
accuracy: 0.7375 - val_loss: 0.5755 - val_accuracy: 0.8000
Epoch 334/500
80/80 [==============================] - 0s 2ms/step - loss: 0.5524 - accuracy:
0.8125 - val_loss: 0.5809 - val_accuracy: 0.7000
Epoch 335/500
80/80 [==============================] - 0s 504us/step - loss: 0.5574 -
accuracy: 0.7750 - val_loss: 0.5892 - val_accuracy: 0.8000
Epoch 336/500
80/80 [==============================] - 0s 480us/step - loss: 0.5526 -
accuracy: 0.7125 - val_loss: 0.5741 - val_accuracy: 0.8000
Epoch 337/500
80/80 [==============================] - 0s 448us/step - loss: 0.5504 -
accuracy: 0.7500 - val_loss: 0.5463 - val_accuracy: 0.7000
```

```
Epoch 338/500
80/80 [==============================] - 0s 524us/step - loss: 0.5629 -
accuracy: 0.7625 - val_loss: 0.5981 - val_accuracy: 0.6000
Epoch 339/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5502 - accuracy:
0.7375 - val_loss: 0.5837 - val_accuracy: 0.8000
Epoch 340/500
80/80 [==============================] - 0s 505us/step - loss: 0.5526 -
accuracy: 0.7375 - val_loss: 0.5629 - val_accuracy: 0.7000
Epoch 341/500
80/80 [==============================] - 0s 389us/step - loss: 0.5524 -
accuracy: 0.6875 - val_loss: 0.6067 - val_accuracy: 0.6000
Epoch 342/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5551 - accuracy:
0.7000 - val_loss: 0.6004 - val_accuracy: 0.6000
Epoch 343/500
80/80 [==============================] - 0s 604us/step - loss: 0.5518 -
accuracy: 0.7375 - val_loss: 0.5759 - val_accuracy: 0.8000
Epoch 344/500
80/80 [==============================] - 0s 446us/step - loss: 0.5522 -
accuracy: 0.7500 - val_loss: 0.5691 - val_accuracy: 0.8000
Epoch 345/500
80/80 [==============================] - 0s 427us/step - loss: 0.5511 -
accuracy: 0.7500 - val_loss: 0.6027 - val_accuracy: 0.6000
Epoch 346/500
80/80 [==============================] - 0s 327us/step - loss: 0.5530 -
accuracy: 0.7125 - val_loss: 0.5959 - val_accuracy: 0.6000
Epoch 347/500
80/80 [==============================] - 0s 925us/step - loss: 0.5471 -
accuracy: 0.7625 - val_loss: 0.5658 - val_accuracy: 0.7000
Epoch 348/500
80/80 [==============================] - 0s 528us/step - loss: 0.5451 -
accuracy: 0.8125 - val_loss: 0.5475 - val_accuracy: 0.7000
Epoch 349/500
80/80 [==============================] - 0s 567us/step - loss: 0.5456 -
accuracy: 0.7625 - val_loss: 0.5558 - val_accuracy: 0.7000
Epoch 350/500
80/80 [==============================] - 0s 335us/step - loss: 0.5412 -
accuracy: 0.8000 - val_loss: 0.5934 - val_accuracy: 0.6000
Epoch 351/500
80/80 [==============================] - 0s 354us/step - loss: 0.5520 -
accuracy: 0.7125 - val_loss: 0.6049 - val_accuracy: 0.6000
Epoch 352/500
80/80 [==============================] - 0s 344us/step - loss: 0.5525 -
accuracy: 0.7125 - val_loss: 0.5638 - val_accuracy: 0.8000
Epoch 353/500
80/80 [==============================] - 0s 412us/step - loss: 0.5416 -
accuracy: 0.7875 - val_loss: 0.5920 - val_accuracy: 0.7000
```

```
Epoch 354/500
80/80 [==============================] - 0s 740us/step - loss: 0.5423 -
accuracy: 0.7500 - val_loss: 0.5613 - val_accuracy: 0.8000
Epoch 355/500
80/80 [==============================] - 0s 623us/step - loss: 0.5485 -
accuracy: 0.7625 - val_loss: 0.5477 - val_accuracy: 0.7000
Epoch 356/500
80/80 [==============================] - 0s 624us/step - loss: 0.5412 -
accuracy: 0.7875 - val_loss: 0.6015 - val_accuracy: 0.6000
Epoch 357/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5455 - accuracy:
0.7250 - val_loss: 0.6088 - val_accuracy: 0.6000
Epoch 358/500
80/80 [==============================] - 0s 475us/step - loss: 0.5442 -
accuracy: 0.7500 - val_loss: 0.5527 - val_accuracy: 0.8000
Epoch 359/500
80/80 [==============================] - 0s 388us/step - loss: 0.5465 -
accuracy: 0.7500 - val_loss: 0.5416 - val_accuracy: 0.7000
Epoch 360/500
80/80 [==============================] - 0s 316us/step - loss: 0.5424 -
accuracy: 0.7625 - val_loss: 0.6146 - val_accuracy: 0.6000
Epoch 361/500
80/80 [==============================] - 0s 343us/step - loss: 0.5472 -
accuracy: 0.7125 - val_loss: 0.6014 - val_accuracy: 0.6000
Epoch 362/500
80/80 [==============================] - 0s 312us/step - loss: 0.5429 -
accuracy: 0.7375 - val_loss: 0.5399 - val_accuracy: 0.7000
Epoch 363/500
80/80 [==============================] - 0s 416us/step - loss: 0.5406 -
accuracy: 0.7250 - val_loss: 0.5416 - val_accuracy: 0.7000
Epoch 364/500
80/80 [==============================] - 0s 491us/step - loss: 0.5389 -
accuracy: 0.7625 - val_loss: 0.5793 - val_accuracy: 0.8000
Epoch 365/500
80/80 [==============================] - 0s 389us/step - loss: 0.5425 -
accuracy: 0.7500 - val_loss: 0.5638 - val_accuracy: 0.8000
Epoch 366/500
80/80 [==============================] - 0s 882us/step - loss: 0.5411 -
accuracy: 0.7625 - val_loss: 0.5776 - val_accuracy: 0.8000
Epoch 367/500
80/80 [==============================] - 0s 368us/step - loss: 0.5419 -
accuracy: 0.7250 - val_loss: 0.5765 - val_accuracy: 0.8000
Epoch 368/500
80/80 [==============================] - 0s 776us/step - loss: 0.5334 -
accuracy: 0.7625 - val_loss: 0.5310 - val_accuracy: 0.7000
Epoch 369/500
80/80 [==============================] - 0s 475us/step - loss: 0.5592 -
accuracy: 0.6875 - val_loss: 0.5255 - val_accuracy: 0.7000
```

```
Epoch 370/500
80/80 [==============================] - 0s 406us/step - loss: 0.5466 -
accuracy: 0.6750 - val_loss: 0.6215 - val_accuracy: 0.6000
Epoch 371/500
80/80 [==============================] - 0s 441us/step - loss: 0.5528 -
accuracy: 0.6750 - val_loss: 0.6231 - val_accuracy: 0.6000
Epoch 372/500
80/80 [==============================] - 0s 376us/step - loss: 0.5435 -
accuracy: 0.7125 - val_loss: 0.5475 - val_accuracy: 0.7000
Epoch 373/500
80/80 [==============================] - 0s 376us/step - loss: 0.5381 -
accuracy: 0.7625 - val_loss: 0.5145 - val_accuracy: 0.7000
Epoch 374/500
80/80 [==============================] - 0s 335us/step - loss: 0.5466 -
accuracy: 0.7125 - val_loss: 0.5651 - val_accuracy: 0.8000
Epoch 375/500
80/80 [==============================] - 0s 325us/step - loss: 0.5375 -
accuracy: 0.7625 - val_loss: 0.5737 - val_accuracy: 0.8000
Epoch 376/500
80/80 [==============================] - 0s 308us/step - loss: 0.5356 -
accuracy: 0.7500 - val_loss: 0.5528 - val_accuracy: 0.8000
Epoch 377/500
80/80 [==============================] - 0s 431us/step - loss: 0.5429 -
accuracy: 0.8000 - val_loss: 0.5457 - val_accuracy: 0.7000
Epoch 378/500
80/80 [==============================] - 0s 421us/step - loss: 0.5341 -
accuracy: 0.7500 - val_loss: 0.6090 - val_accuracy: 0.6000
Epoch 379/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5456 - accuracy:
0.7125 - val_loss: 0.6033 - val_accuracy: 0.6000
Epoch 380/500
80/80 [==============================] - 0s 503us/step - loss: 0.5394 -
accuracy: 0.7250 - val_loss: 0.5403 - val_accuracy: 0.7000
Epoch 381/500
80/80 [==============================] - 0s 310us/step - loss: 0.5467 -
accuracy: 0.7125 - val_loss: 0.5410 - val_accuracy: 0.7000
Epoch 382/500
80/80 [==============================] - 0s 669us/step - loss: 0.5380 -
accuracy: 0.7875 - val_loss: 0.5694 - val_accuracy: 0.8000
Epoch 383/500
80/80 [==============================] - 0s 341us/step - loss: 0.5344 -
accuracy: 0.7625 - val_loss: 0.5644 - val_accuracy: 0.8000
Epoch 384/500
80/80 [==============================] - 0s 366us/step - loss: 0.5335 -
accuracy: 0.7750 - val_loss: 0.5571 - val_accuracy: 0.8000
Epoch 385/500
80/80 [==============================] - 0s 563us/step - loss: 0.5315 -
accuracy: 0.8000 - val_loss: 0.5543 - val_accuracy: 0.8000
```

```
Epoch 386/500
80/80 [==============================] - 0s 388us/step - loss: 0.5314 -
accuracy: 0.7875 - val_loss: 0.5517 - val_accuracy: 0.8000
Epoch 387/500
80/80 [==============================] - 0s 398us/step - loss: 0.5346 -
accuracy: 0.7875 - val_loss: 0.5653 - val_accuracy: 0.8000
Epoch 388/500
80/80 [==============================] - 0s 719us/step - loss: 0.5338 -
accuracy: 0.7375 - val_loss: 0.5961 - val_accuracy: 0.6000
Epoch 389/500
80/80 [==============================] - 0s 2ms/step - loss: 0.5452 - accuracy:
0.7250 - val_loss: 0.5522 - val_accuracy: 0.8000
Epoch 390/500
80/80 [==============================] - 0s 335us/step - loss: 0.5295 -
accuracy: 0.7875 - val_loss: 0.5491 - val_accuracy: 0.8000
Epoch 391/500
80/80 [==============================] - 0s 390us/step - loss: 0.5286 -
accuracy: 0.8000 - val_loss: 0.5346 - val_accuracy: 0.8000
Epoch 392/500
80/80 [==============================] - 0s 396us/step - loss: 0.5298 -
accuracy: 0.7875 - val_loss: 0.5555 - val_accuracy: 0.8000
Epoch 393/500
80/80 [==============================] - 0s 422us/step - loss: 0.5302 -
accuracy: 0.7875 - val_loss: 0.5417 - val_accuracy: 0.8000
Epoch 394/500
80/80 [==============================] - 0s 481us/step - loss: 0.5260 -
accuracy: 0.8000 - val_loss: 0.5646 - val_accuracy: 0.8000
Epoch 395/500
80/80 [==============================] - 0s 408us/step - loss: 0.5311 -
accuracy: 0.7500 - val_loss: 0.5746 - val_accuracy: 0.8000
Epoch 396/500
80/80 [==============================] - 0s 294us/step - loss: 0.5348 -
accuracy: 0.7750 - val_loss: 0.5186 - val_accuracy: 0.7000
Epoch 397/500
80/80 [==============================] - 0s 406us/step - loss: 0.5284 -
accuracy: 0.7500 - val_loss: 0.5336 - val_accuracy: 0.8000
Epoch 398/500
80/80 [==============================] - 0s 277us/step - loss: 0.5268 -
accuracy: 0.8000 - val_loss: 0.5562 - val_accuracy: 0.8000
Epoch 399/500
80/80 [==============================] - 0s 255us/step - loss: 0.5275 -
accuracy: 0.7625 - val_loss: 0.5487 - val_accuracy: 0.8000
Epoch 400/500
80/80 [==============================] - 0s 508us/step - loss: 0.5259 -
accuracy: 0.8125 - val_loss: 0.5604 - val_accuracy: 0.8000
Epoch 401/500
80/80 [==============================] - 0s 420us/step - loss: 0.5304 -
accuracy: 0.7500 - val_loss: 0.5440 - val_accuracy: 0.8000
```

```
Epoch 402/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5267 - accuracy:
0.7625 - val_loss: 0.5587 - val_accuracy: 0.8000
Epoch 403/500
80/80 [==============================] - 0s 340us/step - loss: 0.5286 -
accuracy: 0.7500 - val_loss: 0.5388 - val_accuracy: 0.8000
Epoch 404/500
80/80 [==============================] - 0s 382us/step - loss: 0.5229 -
accuracy: 0.8000 - val_loss: 0.5029 - val_accuracy: 0.7000
Epoch 405/500
80/80 [==============================] - 0s 356us/step - loss: 0.5295 -
accuracy: 0.7500 - val_loss: 0.5230 - val_accuracy: 0.8000
Epoch 406/500
80/80 [==============================] - 0s 344us/step - loss: 0.5212 -
accuracy: 0.8000 - val_loss: 0.5943 - val_accuracy: 0.6000
Epoch 407/500
80/80 [==============================] - 0s 349us/step - loss: 0.5315 -
accuracy: 0.7250 - val_loss: 0.5732 - val_accuracy: 0.8000
Epoch 408/500
80/80 [==============================] - 0s 667us/step - loss: 0.5242 -
accuracy: 0.7625 - val_loss: 0.5028 - val_accuracy: 0.7000
Epoch 409/500
80/80 [==============================] - 0s 864us/step - loss: 0.5295 -
accuracy: 0.7250 - val_loss: 0.5198 - val_accuracy: 0.8000
Epoch 410/500
80/80 [==============================] - 0s 441us/step - loss: 0.5211 -
accuracy: 0.7875 - val_loss: 0.5922 - val_accuracy: 0.6000
Epoch 411/500
80/80 [==============================] - 0s 375us/step - loss: 0.5306 -
accuracy: 0.7250 - val_loss: 0.5781 - val_accuracy: 0.8000
Epoch 412/500
80/80 [==============================] - 0s 435us/step - loss: 0.5221 -
accuracy: 0.7500 - val_loss: 0.5044 - val_accuracy: 0.7000
Epoch 413/500
80/80 [==============================] - 0s 814us/step - loss: 0.5473 -
accuracy: 0.7125 - val_loss: 0.4872 - val_accuracy: 0.8000
Epoch 414/500
80/80 [==============================] - 0s 510us/step - loss: 0.5305 -
accuracy: 0.7625 - val_loss: 0.6124 - val_accuracy: 0.6000
Epoch 415/500
80/80 [==============================] - 0s 779us/step - loss: 0.5395 -
accuracy: 0.7125 - val_loss: 0.6350 - val_accuracy: 0.5000
Epoch 416/500
80/80 [==============================] - 0s 338us/step - loss: 0.5401 -
accuracy: 0.7125 - val_loss: 0.6029 - val_accuracy: 0.6000
Epoch 417/500
80/80 [==============================] - 0s 405us/step - loss: 0.5344 -
accuracy: 0.7250 - val_loss: 0.5440 - val_accuracy: 0.8000
```

```
Epoch 418/500
80/80 [==============================] - 0s 386us/step - loss: 0.5268 -
accuracy: 0.8000 - val_loss: 0.5357 - val_accuracy: 0.7000
Epoch 419/500
80/80 [==============================] - 0s 416us/step - loss: 0.5277 -
accuracy: 0.7875 - val_loss: 0.5271 - val_accuracy: 0.8000
Epoch 420/500
80/80 [==============================] - 0s 558us/step - loss: 0.5242 -
accuracy: 0.8000 - val_loss: 0.5176 - val_accuracy: 0.8000
Epoch 421/500
80/80 [==============================] - 0s 573us/step - loss: 0.5196 -
accuracy: 0.8000 - val_loss: 0.5388 - val_accuracy: 0.8000
Epoch 422/500
80/80 [==============================] - 0s 516us/step - loss: 0.5209 -
accuracy: 0.7750 - val_loss: 0.5535 - val_accuracy: 0.8000
Epoch 423/500
80/80 [==============================] - 0s 547us/step - loss: 0.5216 -
accuracy: 0.8000 - val_loss: 0.5308 - val_accuracy: 0.8000
Epoch 424/500
80/80 [==============================] - 0s 945us/step - loss: 0.5326 -
accuracy: 0.7750 - val_loss: 0.5195 - val_accuracy: 0.8000
Epoch 425/500
80/80 [==============================] - 0s 952us/step - loss: 0.5216 -
accuracy: 0.7875 - val_loss: 0.5753 - val_accuracy: 0.8000
Epoch 426/500
80/80 [==============================] - 0s 586us/step - loss: 0.5221 -
accuracy: 0.7500 - val_loss: 0.5404 - val_accuracy: 0.8000
Epoch 427/500
80/80 [==============================] - 0s 363us/step - loss: 0.5196 -
accuracy: 0.7875 - val_loss: 0.5004 - val_accuracy: 0.7000
Epoch 428/500
80/80 [==============================] - 0s 440us/step - loss: 0.5271 -
accuracy: 0.7500 - val_loss: 0.5285 - val_accuracy: 0.8000
Epoch 429/500
80/80 [==============================] - 0s 642us/step - loss: 0.5155 -
accuracy: 0.7875 - val_loss: 0.5944 - val_accuracy: 0.6000
Epoch 430/500
80/80 [==============================] - 0s 544us/step - loss: 0.5295 -
accuracy: 0.7250 - val_loss: 0.5868 - val_accuracy: 0.6000
Epoch 431/500
80/80 [==============================] - 0s 406us/step - loss: 0.5255 -
accuracy: 0.7375 - val_loss: 0.5114 - val_accuracy: 0.8000
Epoch 432/500
80/80 [==============================] - 0s 648us/step - loss: 0.5180 -
accuracy: 0.7750 - val_loss: 0.5101 - val_accuracy: 0.7000
Epoch 433/500
80/80 [==============================] - 0s 711us/step - loss: 0.5234 -
accuracy: 0.7125 - val_loss: 0.5279 - val_accuracy: 0.8000
```

```
Epoch 434/500
80/80 [==============================] - 0s 720us/step - loss: 0.5198 -
accuracy: 0.7750 - val_loss: 0.5631 - val_accuracy: 0.8000
Epoch 435/500
80/80 [==============================] - 0s 417us/step - loss: 0.5192 -
accuracy: 0.7500 - val_loss: 0.5650 - val_accuracy: 0.8000
Epoch 436/500
80/80 [==============================] - 0s 494us/step - loss: 0.5189 -
accuracy: 0.7625 - val_loss: 0.5150 - val_accuracy: 0.8000
Epoch 437/500
80/80 [==============================] - 0s 359us/step - loss: 0.5153 -
accuracy: 0.7750 - val_loss: 0.4987 - val_accuracy: 0.8000
Epoch 438/500
80/80 [==============================] - 0s 337us/step - loss: 0.5154 -
accuracy: 0.7875 - val_loss: 0.5407 - val_accuracy: 0.8000
Epoch 439/500
80/80 [==============================] - 0s 612us/step - loss: 0.5252 -
accuracy: 0.7375 - val_loss: 0.5820 - val_accuracy: 0.8000
Epoch 440/500
80/80 [==============================] - 0s 428us/step - loss: 0.5221 -
accuracy: 0.7375 - val_loss: 0.5166 - val_accuracy: 0.8000
Epoch 441/500
80/80 [==============================] - 0s 671us/step - loss: 0.5164 -
accuracy: 0.7875 - val_loss: 0.5037 - val_accuracy: 0.8000
Epoch 442/500
80/80 [==============================] - 0s 360us/step - loss: 0.5232 -
accuracy: 0.7750 - val_loss: 0.5166 - val_accuracy: 0.8000
Epoch 443/500
80/80 [==============================] - 0s 380us/step - loss: 0.5213 -
accuracy: 0.7750 - val_loss: 0.5091 - val_accuracy: 0.8000
Epoch 444/500
80/80 [==============================] - 0s 585us/step - loss: 0.5172 -
accuracy: 0.7375 - val_loss: 0.5716 - val_accuracy: 0.8000
Epoch 445/500
80/80 [==============================] - 0s 412us/step - loss: 0.5196 -
accuracy: 0.7500 - val_loss: 0.5553 - val_accuracy: 0.8000
Epoch 446/500
80/80 [==============================] - 0s 1ms/step - loss: 0.5204 - accuracy:
0.7500 - val_loss: 0.5055 - val_accuracy: 0.8000
Epoch 447/500
80/80 [==============================] - 0s 378us/step - loss: 0.5175 -
accuracy: 0.7500 - val_loss: 0.5305 - val_accuracy: 0.8000
Epoch 448/500
80/80 [==============================] - 0s 543us/step - loss: 0.5131 -
accuracy: 0.7750 - val_loss: 0.5801 - val_accuracy: 0.7000
Epoch 449/500
80/80 [==============================] - 0s 363us/step - loss: 0.5250 -
accuracy: 0.7250 - val_loss: 0.5636 - val_accuracy: 0.8000
```

```
Epoch 450/500
80/80 [==============================] - 0s 337us/step - loss: 0.5204 -
accuracy: 0.7750 - val_loss: 0.5007 - val_accuracy: 0.8000
Epoch 451/500
80/80 [==============================] - 0s 372us/step - loss: 0.5266 -
accuracy: 0.7125 - val_loss: 0.5058 - val_accuracy: 0.8000
Epoch 452/500
80/80 [==============================] - 0s 710us/step - loss: 0.5087 -
accuracy: 0.8125 - val_loss: 0.5709 - val_accuracy: 0.8000
Epoch 453/500
80/80 [==============================] - 0s 373us/step - loss: 0.5192 -
accuracy: 0.7375 - val_loss: 0.5698 - val_accuracy: 0.8000
Epoch 454/500
80/80 [==============================] - 0s 322us/step - loss: 0.5146 -
accuracy: 0.7375 - val_loss: 0.5024 - val_accuracy: 0.8000
Epoch 455/500
80/80 [==============================] - 0s 295us/step - loss: 0.5199 -
accuracy: 0.7875 - val_loss: 0.4624 - val_accuracy: 0.8000
Epoch 456/500
80/80 [==============================] - 0s 286us/step - loss: 0.5202 -
accuracy: 0.7375 - val_loss: 0.5463 - val_accuracy: 0.8000
Epoch 457/500
80/80 [==============================] - 0s 297us/step - loss: 0.5155 -
accuracy: 0.7375 - val_loss: 0.6334 - val_accuracy: 0.5000
Epoch 458/500
80/80 [==============================] - 0s 442us/step - loss: 0.5359 -
accuracy: 0.7000 - val_loss: 0.6191 - val_accuracy: 0.6000
Epoch 459/500
80/80 [==============================] - 0s 535us/step - loss: 0.5234 -
accuracy: 0.7125 - val_loss: 0.5152 - val_accuracy: 0.8000
Epoch 460/500
80/80 [==============================] - 0s 443us/step - loss: 0.5131 -
accuracy: 0.7375 - val_loss: 0.4661 - val_accuracy: 0.8000
Epoch 461/500
80/80 [==============================] - 0s 387us/step - loss: 0.5183 -
accuracy: 0.7625 - val_loss: 0.5305 - val_accuracy: 0.8000
Epoch 462/500
80/80 [==============================] - 0s 401us/step - loss: 0.5087 -
accuracy: 0.7750 - val_loss: 0.5807 - val_accuracy: 0.7000
Epoch 463/500
80/80 [==============================] - 0s 291us/step - loss: 0.5208 -
accuracy: 0.7500 - val_loss: 0.5668 - val_accuracy: 0.8000
Epoch 464/500
80/80 [==============================] - 0s 397us/step - loss: 0.5221 -
accuracy: 0.7250 - val_loss: 0.5173 - val_accuracy: 0.8000
Epoch 465/500
80/80 [==============================] - 0s 636us/step - loss: 0.5124 -
accuracy: 0.7625 - val_loss: 0.5305 - val_accuracy: 0.8000
```

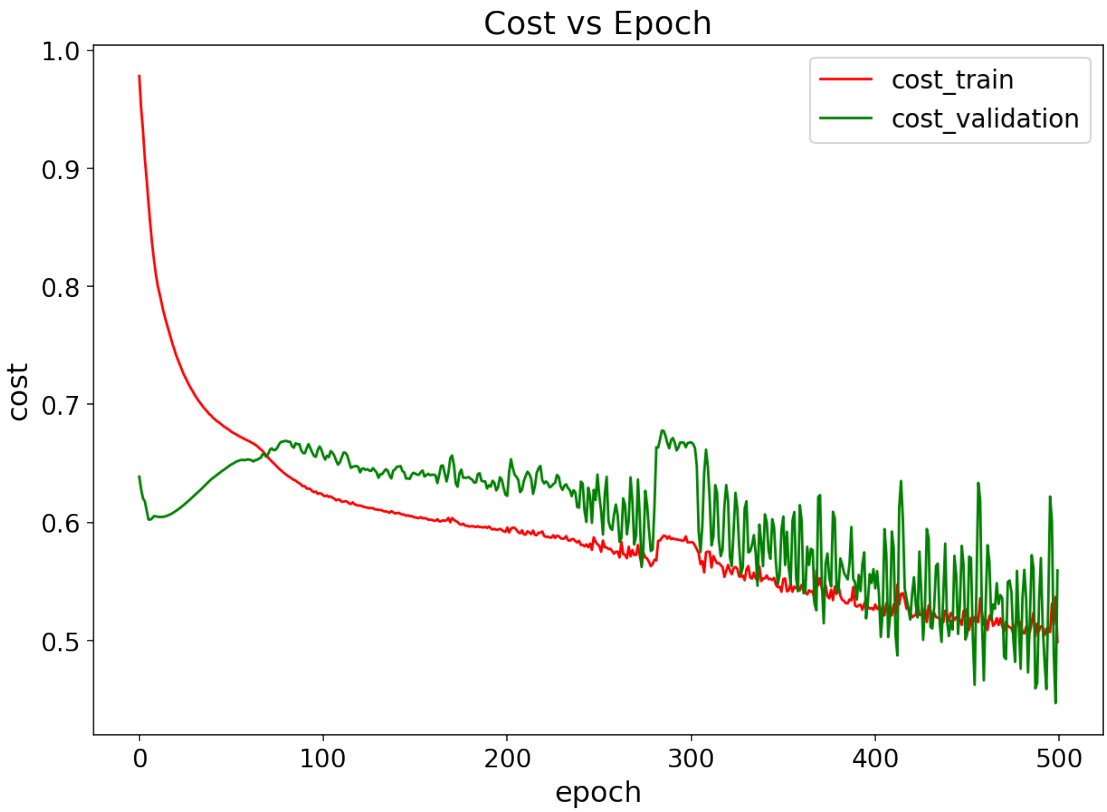
```
Epoch 466/500
80/80 [==============================] - 0s 545us/step - loss: 0.5141 -
accuracy: 0.7875 - val_loss: 0.5267 - val_accuracy: 0.8000
Epoch 467/500
80/80 [==============================] - 0s 360us/step - loss: 0.5188 -
accuracy: 0.7750 - val_loss: 0.5485 - val_accuracy: 0.8000
Epoch 468/500
80/80 [==============================] - 0s 335us/step - loss: 0.5130 -
accuracy: 0.7625 - val_loss: 0.5308 - val_accuracy: 0.8000
Epoch 469/500
80/80 [==============================] - 0s 281us/step - loss: 0.5187 -
accuracy: 0.7375 - val_loss: 0.5384 - val_accuracy: 0.8000
Epoch 470/500
80/80 [==============================] - 0s 298us/step - loss: 0.5119 -
accuracy: 0.7500 - val_loss: 0.5359 - val_accuracy: 0.8000
Epoch 471/500
80/80 [==============================] - 0s 269us/step - loss: 0.5075 -
accuracy: 0.7375 - val_loss: 0.4863 - val_accuracy: 0.7000
Epoch 472/500
80/80 [==============================] - 0s 358us/step - loss: 0.5158 -
accuracy: 0.7500 - val_loss: 0.4842 - val_accuracy: 0.8000
Epoch 473/500
80/80 [==============================] - 0s 393us/step - loss: 0.5124 -
accuracy: 0.7750 - val_loss: 0.5495 - val_accuracy: 0.8000
Epoch 474/500
80/80 [==============================] - 0s 619us/step - loss: 0.5112 -
accuracy: 0.7375 - val_loss: 0.5508 - val_accuracy: 0.8000
Epoch 475/500
80/80 [==============================] - 0s 637us/step - loss: 0.5103 -
accuracy: 0.7500 - val_loss: 0.5409 - val_accuracy: 0.8000
Epoch 476/500
80/80 [==============================] - 0s 403us/step - loss: 0.5057 -
accuracy: 0.7750 - val_loss: 0.5014 - val_accuracy: 0.8000
Epoch 477/500
80/80 [==============================] - 0s 314us/step - loss: 0.5129 -
accuracy: 0.7750 - val_loss: 0.4818 - val_accuracy: 0.8000
Epoch 478/500
80/80 [==============================] - 0s 572us/step - loss: 0.5179 -
accuracy: 0.7500 - val_loss: 0.5589 - val_accuracy: 0.8000
Epoch 479/500
80/80 [==============================] - 0s 541us/step - loss: 0.5122 -
accuracy: 0.7375 - val_loss: 0.5137 - val_accuracy: 0.8000
Epoch 480/500
80/80 [==============================] - 0s 384us/step - loss: 0.5154 -
accuracy: 0.7625 - val_loss: 0.4760 - val_accuracy: 0.7000
Epoch 481/500
80/80 [==============================] - 0s 336us/step - loss: 0.5068 -
accuracy: 0.7750 - val_loss: 0.5356 - val_accuracy: 0.8000
```

```
Epoch 482/500
80/80 [==============================] - 0s 318us/step - loss: 0.5059 -
accuracy: 0.7500 - val_loss: 0.5598 - val_accuracy: 0.8000
Epoch 483/500
80/80 [==============================] - 0s 323us/step - loss: 0.5118 -
accuracy: 0.7375 - val_loss: 0.5354 - val_accuracy: 0.8000
Epoch 484/500
80/80 [==============================] - 0s 307us/step - loss: 0.5109 -
accuracy: 0.7500 - val_loss: 0.4728 - val_accuracy: 0.8000
Epoch 485/500
80/80 [==============================] - 0s 829us/step - loss: 0.5067 -
accuracy: 0.7875 - val_loss: 0.5169 - val_accuracy: 0.8000
Epoch 486/500
80/80 [==============================] - 0s 394us/step - loss: 0.5153 -
accuracy: 0.7625 - val_loss: 0.5722 - val_accuracy: 0.7000
Epoch 487/500
80/80 [==============================] - 0s 336us/step - loss: 0.5237 -
accuracy: 0.7250 - val_loss: 0.5609 - val_accuracy: 0.8000
Epoch 488/500
80/80 [==============================] - 0s 373us/step - loss: 0.5035 -
accuracy: 0.7625 - val_loss: 0.4594 - val_accuracy: 0.7000
Epoch 489/500
80/80 [==============================] - 0s 326us/step - loss: 0.5143 -
accuracy: 0.7625 - val_loss: 0.4641 - val_accuracy: 0.7000
Epoch 490/500
80/80 [==============================] - 0s 349us/step - loss: 0.5081 -
accuracy: 0.7875 - val_loss: 0.5334 - val_accuracy: 0.8000
Epoch 491/500
80/80 [==============================] - 0s 357us/step - loss: 0.5119 -
accuracy: 0.7625 - val_loss: 0.5697 - val_accuracy: 0.8000
Epoch 492/500
80/80 [==============================] - 0s 713us/step - loss: 0.5114 -
accuracy: 0.7750 - val_loss: 0.5118 - val_accuracy: 0.8000
Epoch 493/500
80/80 [==============================] - 0s 278us/step - loss: 0.5043 -
accuracy: 0.7750 - val_loss: 0.4822 - val_accuracy: 0.8000
Epoch 494/500
80/80 [==============================] - 0s 263us/step - loss: 0.5100 -
accuracy: 0.7500 - val_loss: 0.4587 - val_accuracy: 0.7000
Epoch 495/500
80/80 [==============================] - 0s 248us/step - loss: 0.5105 -
accuracy: 0.7500 - val_loss: 0.5307 - val_accuracy: 0.8000
Epoch 496/500
80/80 [==============================] - 0s 259us/step - loss: 0.5070 -
accuracy: 0.7500 - val_loss: 0.6219 - val_accuracy: 0.5000
Epoch 497/500
80/80 [==============================] - 0s 223us/step - loss: 0.5312 -
accuracy: 0.6875 - val_loss: 0.6013 - val_accuracy: 0.7000
```

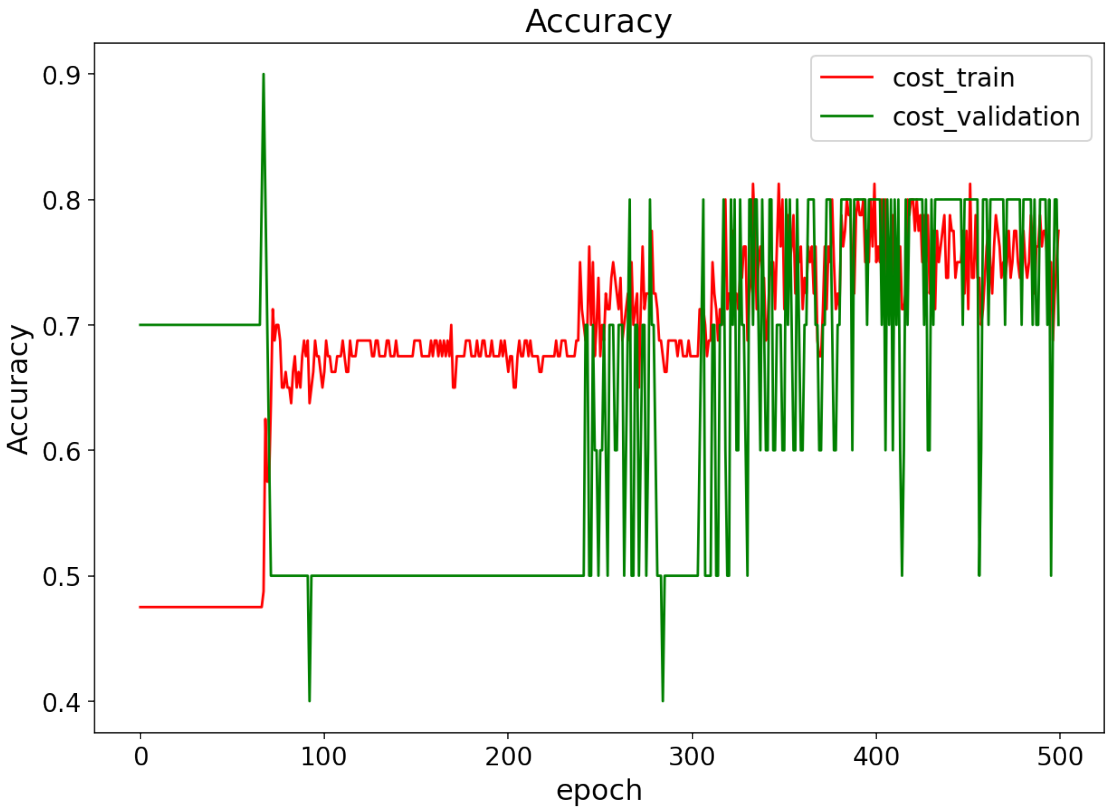
```
Epoch 498/500
80/80 [==============================] - 0s 280us/step - loss: 0.5135 -
accuracy: 0.7250 - val_loss: 0.4887 - val_accuracy: 0.8000
Epoch 499/500
80/80 [==============================] - 0s 341us/step - loss: 0.5369 -
accuracy: 0.7500 - val_loss: 0.4469 - val_accuracy: 0.8000
Epoch 500/500
80/80 [==============================] - 0s 490us/step - loss: 0.4986 -
accuracy: 0.7750 - val_loss: 0.5592 - val_accuracy: 0.7000
```

1.6 Development of the network in each epoch

[28]:
```
#Cost function in every epoch
plt.figure(figsize=(10, 7))
color="black"
plt.plot(history.history['loss'], color='red')
plt.plot(history.history['val_loss'], color='green')
plt.title('Cost vs Epoch', color=color)
plt.ylabel('cost',color=color)
plt.xlabel('epoch',color=color)
plt.legend(['cost_train', 'cost_validation'])

plt.tick_params(axis='x', colors=color)
plt.tick_params(axis='y', colors=color)
#plt.savefig("Build.png")
plt.show()
```



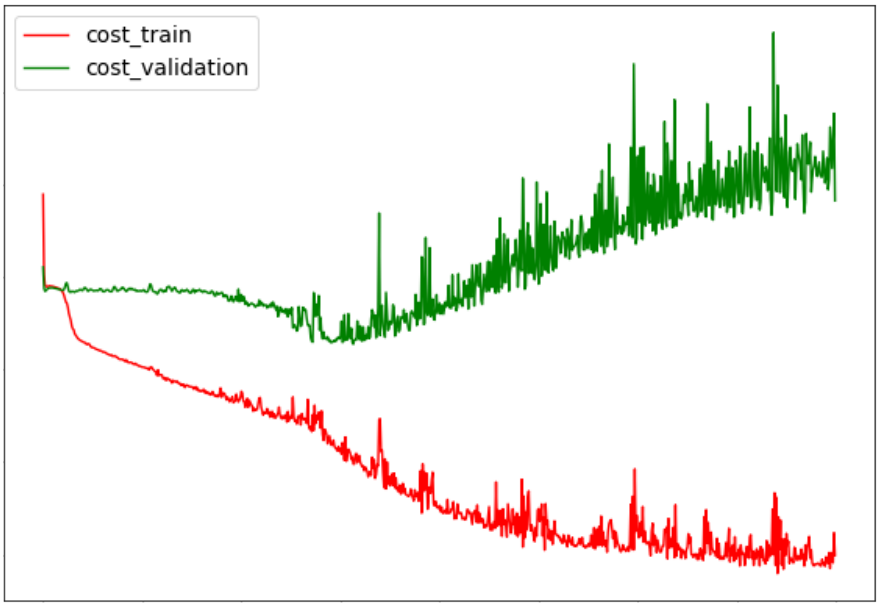
[29]:
```
# Accuracy vs epoch
plt.figure(figsize=(10, 7))

plt.plot(history.history['accuracy'], color='red')
plt.plot(history.history['val_accuracy'], color='green')
plt.title('Accuracy', color=color)
plt.ylabel('Accuracy', color=color)
plt.xlabel('epoch', color=color)
plt.legend(['cost_train', 'cost_validation'])
plt.tick_params(axis='x', colors=color)
plt.tick_params(axis='y', colors=color)
#plt.savefig("Acuracy.png")
plt.show()
```



1.7 Interpretation of the cost vs epoch graphs

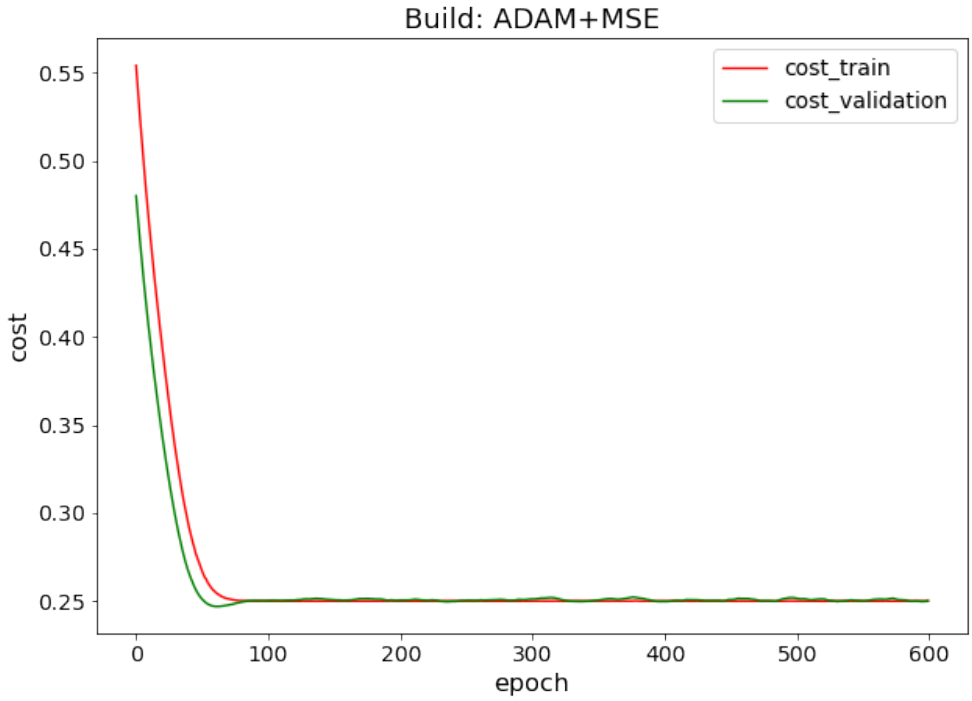
To evaluate the chosen architecture there is a validation group. Now we will see how to know if you chose the right architecture for the fenomena you are trying to predict.

One of the most common mistakes in AI training is over-adjustments. You can notice an overadjustment when the "train" group keeps being optimized but the "validation" group doesn't. In the following graph you can observe an example of an overadjustment:



How to solve an overadjustment: - Simplify the architecture of the algorithm - Add a dropout layer. This is a layer in which every neuron has a chance to disapear. This forcesthe remaining neurons to become more representative. - Reduce the number of epochs in the epoch in which the two lines start to diverge.

In an ideal case the two lines would go down together and in the end of the graph the optimization would start to decrease. If both lines keep decreasing, the algorithm needs more epochs and the results would improve. In the following graph you can observe how the algoritm reaches a local minimum in the cost function after aproximately 70 epochs:



If the final accuracy isn't acceptable or the cost function stops decreasing, we want to reduce the number of epochs, reduce the learning rate or increase the complexity of the neural network.

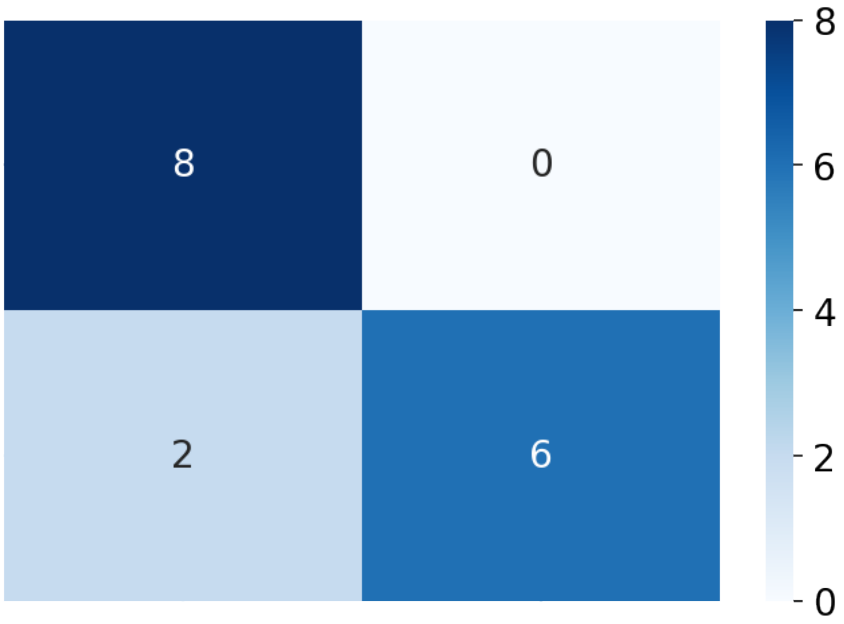
Too much complexity leads to an over-adjustment, while a lack of complexity results in the network not properly adapting to the phenomena. The architecture must be changed until a balance is achived.

1.8 Evaluation

[30]:
```
#The condition of each sample is predicted using the trained network
prediction = model.predict(test_x)
```

[31]:
```
#A confution matrix is made to compare the predicted condition vs the actual␣
 ↪condition
confusion_matrix = pd.crosstab(test_y[:,0], np.rint(prediction)[:,0],␣
 ↪rownames=['Actual'], colnames=['Predicted'])

sn.heatmap(confusion_matrix,␣
 ↪annot=True,cmap='Blues',xticklabels=["Normal","Arrythmia"],yticklabels=["Normal","Arrythmia

plt.tick_params(axis='x', colors='white')
plt.xlabel("Predicted",color="white")
plt.ylabel("Actual",color="white")
plt.tick_params(axis='y', colors='white')
```

```
plt.savefig("confusion.png")
plt.show()
```



[32]:
```
13/16
```

[32]: 0.8125

[ ]:
```

```