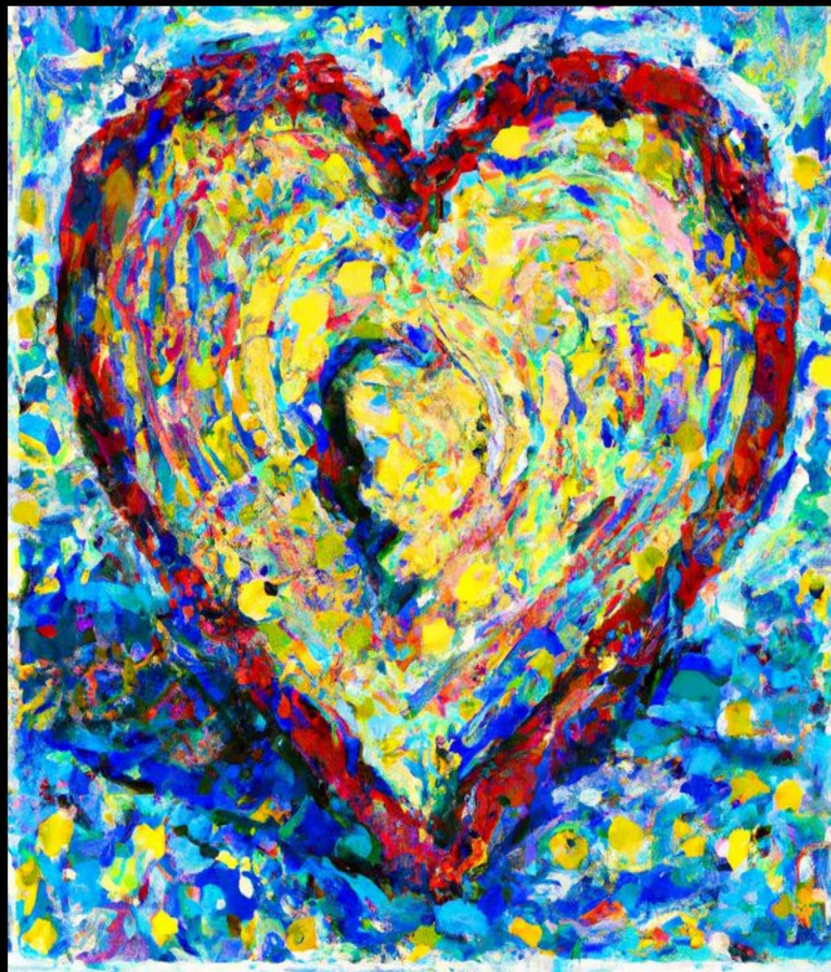
Locating S1 and S2 Sounds: A Machine Learning Approach

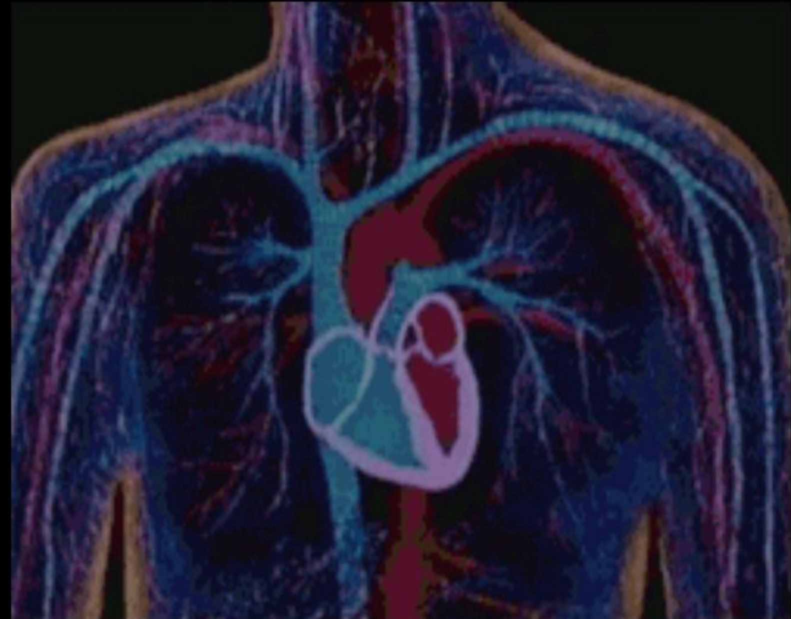
Introduction

Heart sound segmentation using machine learning is a powerful tool for identifying S1 and S2 sounds in audio data.



The Identification of S1 and S2 Sounds

The S1 and S2 sounds correspond to the closure of the mitral and tricuspid valves, respectively, and the rhythm of those sounds can provide important information about a patient's heart function and about acute malfunctions.



The Benefits of Automated S1 and S2 Sound Analysis

The identification of S1 and S2 sounds in audio data is critical for the diagnosis and management of heart conditions.

Automating this process can reduce the need for manual interpretation of audio recordings, which can be time-consuming and subject to human error.

Real-time analysis might also give warnings or even predict trouble.

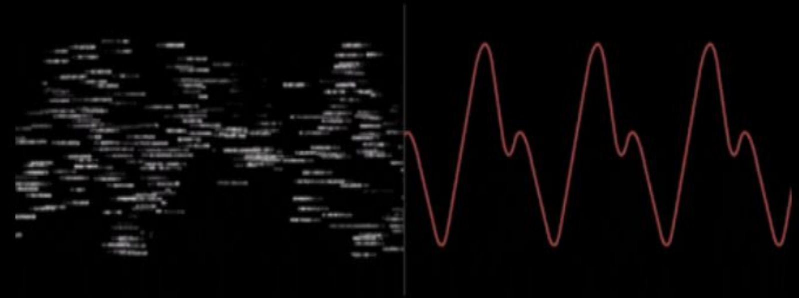
Problem Description

The main challenge is to develop a method for accurately identifying and locating the S1 (lub) and S2 (dub) sounds within audio data of normal heartbeats. This includes segmenting the audio files in both datasets, using labeled data to train the method, and using the learned method to identify and locate these sounds in an unlabeled dataset.

Main Challenges of Locating S1 and S2 Sounds

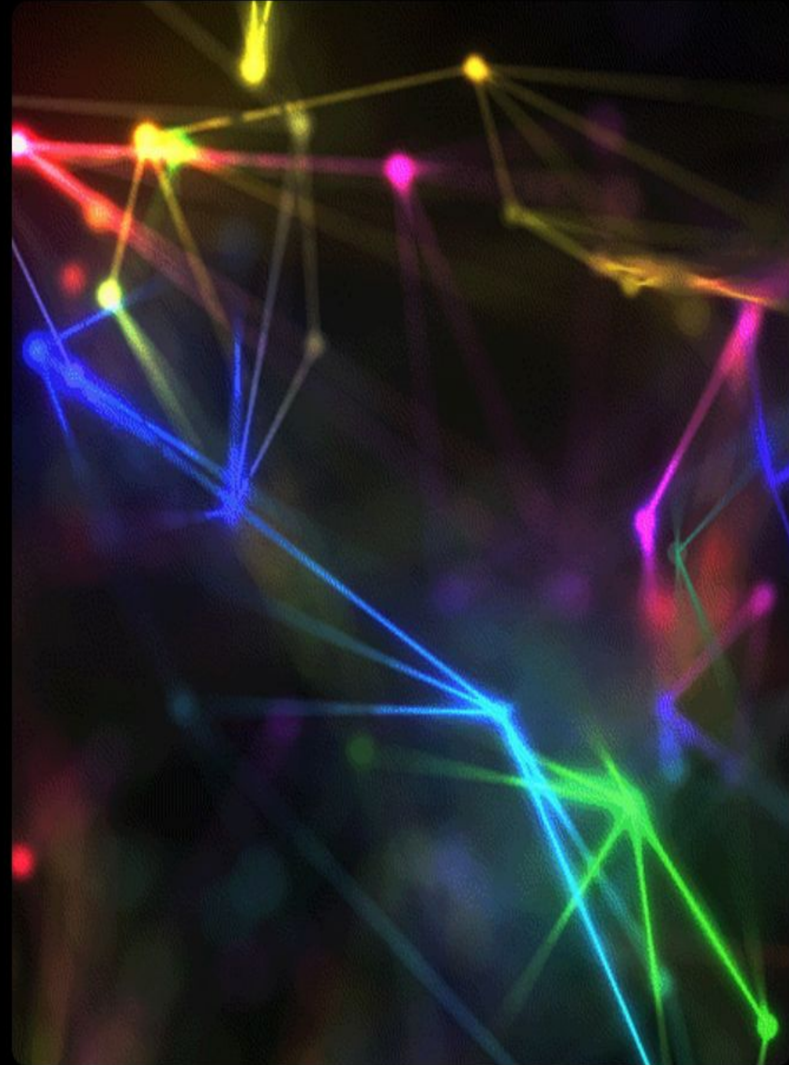
One of the main challenges of this problem is the variability of heart sounds across different individuals, as well as across different heart conditions.

Another challenge is the need for high precision and accuracy in the location of S1 and S2 sounds while there is a lot of noise.



Proposed Solution for Locating S1 and S2 Sounds

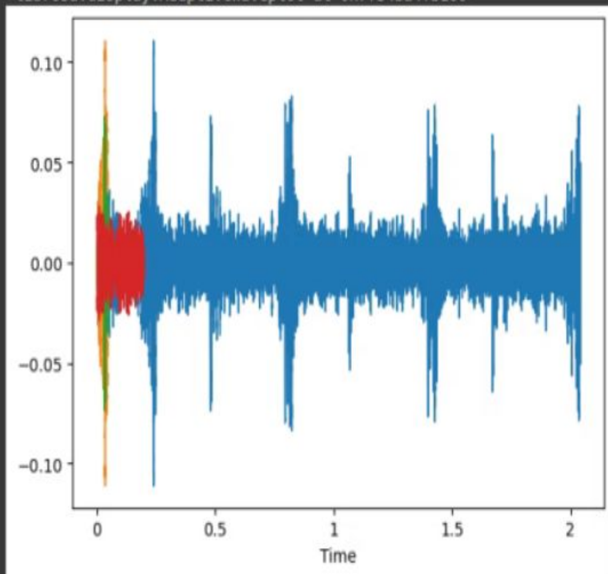
This proposed solution aims to accurately segment audio recordings into individual heartbeats and classify each heartbeat as S1 or S2 sound using machine learning techniques

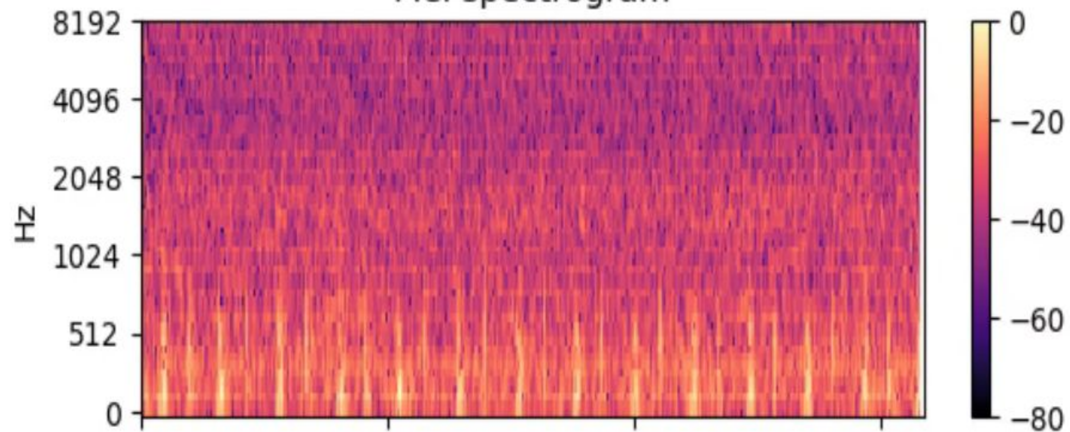


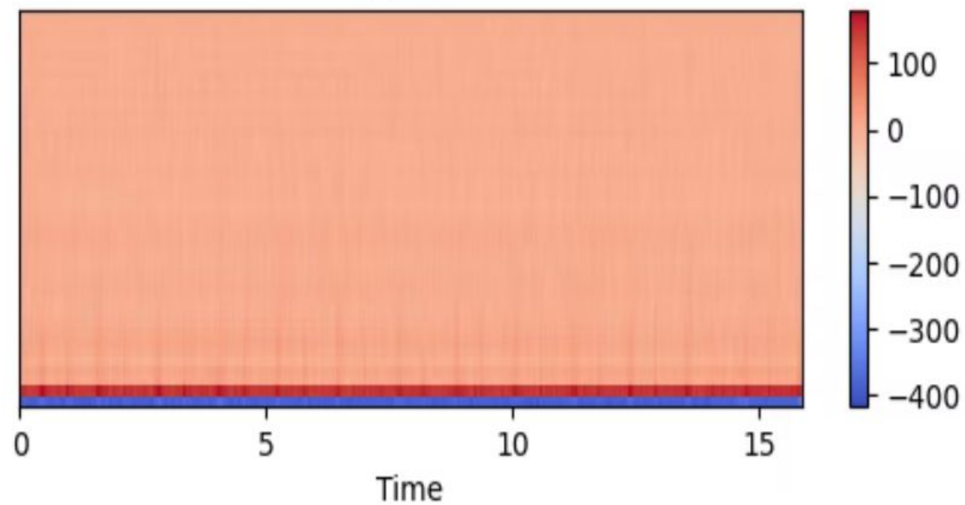
```
temp_value1 = df.iloc[0, 1]
temp_value2 = df.iloc[0, 1+1]

interval = 1000
data1, sampling_rate = librosa.load('Atraining_normal/'+ df.iloc[0, 0].split('.')[0] +'.wav', sr=44100 )
signal = data1[:90000]
S1 = data1[int(temp_value1-interval):int(temp_value1+interval)]
S2 = data1[int(temp_value2-interval):int(temp_value2+interval)]
noise = data1[int(temp_value1+interval):int(temp_value2-interval)]
display.waveshow(signal, sr=sampling_rate)
display.waveshow(S1, sr=sampling_rate)
display.waveshow(S2, sr=sampling_rate)
display.waveshow(noise, sr=sampling_rate)
```

```
<librosa.display.AdaptiveWaveplot at 0x7f34ba4fb100>
```




Mel spectrogram
8192
4096
2048
1024
512
0
Hz
0
−20
−40
−60
−80


MFCC
100
0
−100
−200
−300
−400
0
5
10
15
Time

```python
data_x = []
data_y = []
noise_x = []
noise_y = []

# Loop through the rows and columns in the DataFrame
for i in range(0, df.shape[0]):
   for j in range(1, df.shape[1]-1):

       #ckecks that the cell is emtpy, but does not skip if another error occurs
        if pd.isnull(df.iloc[i, j]):
           continue

        # Load the audio data and retrieve the label and value from the DataFrame
        data, sampling_rate = librosa.load('Atraining_normal/'+ df.iloc[i, 0].split('.')[0] +'.wav', sr=44100 )
        temp_label = df.iloc[:, j].name.split('.')[0]

        #labels given are not centered but are offset by around 250 frames
        offset = 250
        temp_value = df.iloc[i, j] + offset
        window = 0.04 #in seconds
        interval = sampling_rate * window

        # Compute the MFCCs and extract the desired data
        # putting a lowest and highest border makes sure that we don't put in invalid indices
        signal = data[int(max(0,temp_value-interval)):min(int(temp_value+interval),data.shape[0])]
```

```python
#make sure that the next cell is not empty
if not pd.isnull(df.iloc[i, j+1]):
    #BENEFITS OF THIS noise-IMPLETATION: - more noise data
                              #PROBLEMS: - might catch other unlabeld peaks
                              #          - samples don't have the same length
                              #            ->something like np.mean is needed
   #THIS ->  noise = data[int((temp_value + interval )):int(((df.iloc[i, j+1]) - interval))]

   #BENEFITS OF the following noise-IMPLETATION: -same sized samples, less likely to catch other peaks
                                      #PROBLEMS: -less data

    noise = data[int(max(0,temp_value+2*interval)):min(int(temp_value+4*interval),data.shape[0])]
    noise_label = 'Noise'
    #in theory n_fft is chosen to high here -> needs more frames than the interval has.
    #somehow still works better than other options
    noisemf = librosa.feature.mfcc(y=noise, sr=sampling_rate, n_mfcc = 50, center=True, n_fft=2048)

    #no mean, but concatenate too save more data and have some kind of marker for the temporal changes of frequenices
    noisemf_processed = np.concatenate(noisemf.T,axis=0)

#same as with noise
mfccs = librosa.feature.mfcc(y=signal, sr=sampling_rate, n_mfcc = 50, center=True, n_fft=2048)
mfccs_processed = np.concatenate(mfccs.T,axis=0)

data_x.append(mfccs_processed)
data_y.append(temp_label)
noise_x.append(noisemf_processed)
noise_y.append(noise_label)
```

```python
print(features.shape, labels.shape)
```

```
(738, 2) (738,)
```

```python
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(labels)
print(integer_encoded)
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
label = onehot_encoder.fit_transform(integer_encoded)
label
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       ...,
       [1., 0.]
```

```
[1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1
 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 1 2 1 2 1 2 1 2 1 2 1
 2 1 2 1 2 1 1 2 1 2 1 2 1 2 1 2 1 1 2 1 2 1 2 1 2 1 2 1 1 2 1 2 1 2 1 2 1
 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1
 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 1 2 1 2 1 2 1
 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1
 2 1 1 2 1 2 1 2 1 2 1 2 1 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 1 2 1 2 1 2 1 2 1
 2 1 2 1 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
 1 2 1 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 1 2 1 2 1 2 1 2 1 2 1 2 1 1 2 1 2
 1 2 1 2 1 2 1 2 1 2 1 2 1 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

array([[0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       ...,
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.]])
```

```python
import tensorflow as tf
from sklearn.metrics import precision_recall_fscore_support, roc_auc_score
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping

x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2, random_state = 96)

# neural network dimensions
n_dim = x_train.shape[1]
n_classes = y_train.shape[1]
n_hidden_units_1 = n_dim
n_hidden_units_2 = 120 # approx n_dim * 2
n_hidden_units_3 = 120
n_hidden_units_4 = 80 #
n_hidden_units_5 = 59 # half of layer 2

print ("Features:", n_dim, "Classes:", n_classes)
```

```
Features: 2 Classes: 2
```

```python
def create_model(activation_function='relu', optimiser='adam', dropout_rate=0.3):
    model = Sequential()
    # layer 1
    model.add(Dense(n_hidden_units_1, input_dim=n_dim, activation='relu'))
    # layer 2
    model.add(Dense(n_hidden_units_2, activation='sigmoid')) #sigmoid
    model.add(Dropout(dropout_rate))
    # layer 3
    model.add(Dense(n_hidden_units_3, activation='relu'))#tanh
    model.add(Dropout(dropout_rate))
     # layer 4
    model.add(Dense(n_hidden_units_4, activation='sigmoid'))#sigmoid
    model.add(Dropout(dropout_rate))

    model.add(Dense(n_hidden_units_4, activation='relu'))#tanh
    model.add(Dropout(dropout_rate))



    # output layer
    model.add(Dense(n_classes, activation='softmax'))

    model.compile(loss='binary_crossentropy', optimizer=optimiser, metrics=['accuracy'])
    return model

# a stopping function to stop training before we excessively overfit to the training set
earlystop = EarlyStopping(monitor='val_loss', patience=5, verbose=1, mode='auto')

model = create_model()

history = model.fit(x_train, y_train,  epochs=25, batch_size=2, validation_data=(x_test, y_test))
```

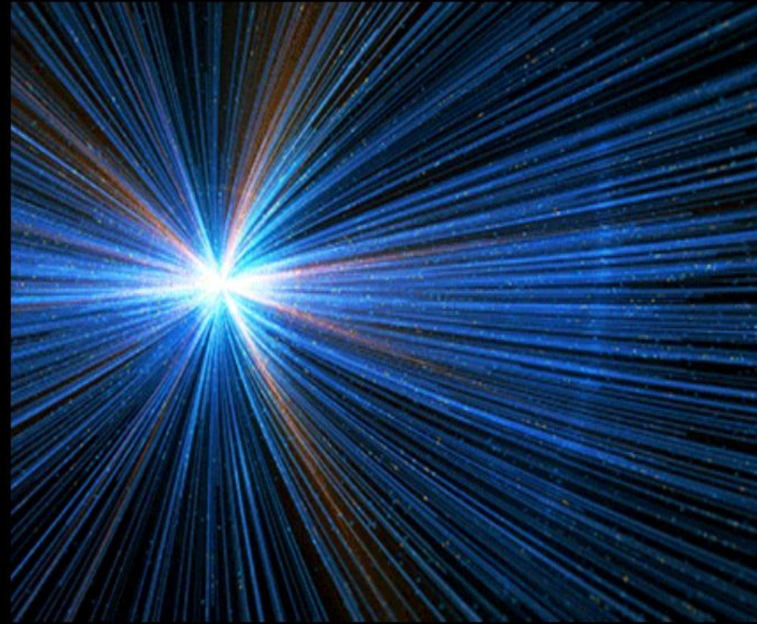
Result - Signal/Noise

```
Epoch 99/100
59/59 [==============================] - 3s 45ms/step - loss: 0.3708 - accuracy: 0.8644 - val_loss: 0.1552 - val_accuracy: 0.9459
Epoch 100/100
59/59 [==============================] - 3s 46ms/step - loss: 0.4149 - accuracy: 0.8390 - val_loss: 0.2706 - val_accuracy: 0.9527
```

Result - S1/S2/Noise

```
Epoch 296/300
125/125 [==============================] - 1s 5ms/step - loss: 0.0908 - accuracy: 0.9550 - val_loss: 0.6202 - val_ac
curacy: 0.8718
Epoch 297/300
125/125 [==============================] - 1s 5ms/step - loss: 0.0591 - accuracy: 0.9662 - val_loss: 0.5587 - val_ac
curacy: 0.8846
Epoch 298/300
125/125 [==============================] - 1s 5ms/step - loss: 0.0683 - accuracy: 0.9598 - val_loss: 0.6511 - val_ac
curacy: 0.9038
Epoch 299/300
125/125 [==============================] - 1s 6ms/step - loss: 0.1905 - accuracy: 0.9084 - val_loss: 0.3575 - val_ac
curacy: 0.8526
Epoch 300/300
125/125 [==============================] - 1s 6ms/step - loss: 0.0705 - accuracy: 0.9630 - val_loss: 0.3683 - val_ac
curacy: 0.8590
```

Conclusion



Q&A

Created by Elias and Mantas