LIP-STUDENTS-22-07

Aplicação de métodos de Machine Learning e Deep Learning na análise de batimentos Cardíacos

Sandra Mamede[1,a]

[1] *Universidade de Coimbra, Coimbra, Portugal*

Supervisor de projeto: Filipe Veloso

October 20, 2022

Abstract. O objetivo deste projeto está relacionado com o estudo de batimentos cardíacos com a utilização de Inteligência Artificial. Tal estudo tem em vista possíveis aplicações médicas como a deteção de arritmia, sopros de coração, cardiomiopatia, entre outros.

1 Introdução

Para poder estudar os sinais sonoros é primeiro necessário fazer o pré-processamento dos mesmos e classificá-los para treinar o modelo.

Usámos então a biblioteca de Python, Librosa, para a análise de ficheiros sonoros onde de seguida os mesmos irão ser usados para treinar uma rede neuronal.

Por vezes não é possível distinguir o tipo de sinal através da observação do gráfico da onda e, por esse motivo, das melhores características a retirar são os coeficientes de Mel (MFCCs).

No processamento de sinais, a amostragem refere-se à redução de um sinal analógico em valores discretos e, a frequência de amostragem corresponde ao número de amostras obtidas num intervalo de tempo fixo. Uma grande frequência de amostragem leva a uma menor perda de informação ao contrário de uma baixa frequência de amostragem, contudo irá a demorar muito mais tempo a correr o programa.

As Transformadas de Fourier permitem converter uma função que dependia da amplitude e do tempo para um outro que dependa da amplitude e das frequências. O que se implementou neste trabalho foi uma outra variante denominada de Short-time Fourier Transform. Esta é implementada na biblioteca do Librosa e o que faz é separar o ficheiro em várias janelas e retira a Transformada de Fourier de cada uma delas.

Algo também a ter presente é a escala de Mel o que, de uma forma simplificada, é um gráfico com forma logarítmica que representa "a capacidade do ouvido humano". Por outras palavras, o ser humano não distingue linearmente a mesma diferença de frequências para sons mais graves e sons mais agudos. A expressão para converter frequências (f-hertz) para mels (m) é:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right). \quad (1)$$

2 Pré-processamento de Sinal

O pré-processamento de sinal é importante bem como a extração de algumas características/informações pois, quando for para treinar o modelo, permite que o mesmo possa classificar a amostra ou até mesmo prever valores. A base que se encontra neste processo é encontrar características que permitem distinguir sinais de outros, sendo os MFCCs a característica favorita escolhida. Necessitamos de seguir os seguintes passos de forma conseguir extrair os MFCCs:

2.1 Separar o sinal em pequenas janelas

Permite amostrar todo o ficheiro áudio em tempos discretos. Assume-se que escalas de tempo extremamente curtas o sinal não se altera. É convencional a sobreposição de pequenos fragmentos de janela adjacente o que irá gerar correlação entre elas.

2.2 Espetro de potência de cada janela

O espetro de potência é uma serie temporal que descreve a distribuição de potencias em componentes de frequência que formam o sinal. De outra forma, o que se faz, é aplicar a Transformada de Fourier em cada janela e faz-se a sua média para obter um espetro de potencia.

2.3 Aplicar filtros de mel aos espetros de potência e soma de energias em cada filtro

Relembrando que o ser humano não consegue distinguir frequências próximas e que tal facto piora com o aumento das frequências, usamos a escala de mel que permite aproximar (mimetiza) a resposta do sistema auditivo humano.

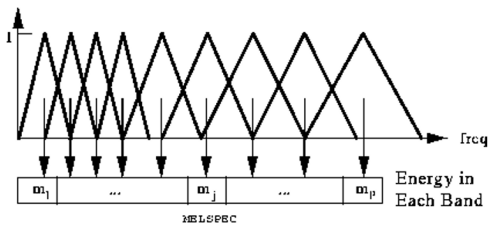

Figure 1. Exemplo de aplicação de um mel-scale filter bank. Imagem retirada de https://towardsdatascience.com/how-to-apply-machine-learning-and-deep-learning-methods-to-audio-analysis-615e286fcbbc

[a] e-mail: uc2019234934@student.uc.pt

Como se pode ver na figura, os filtros de Mel vão alargando com o aumento da frequência. Não importa muito estudar as variações em altas frequências ao contrário das baixas frequências que são mais percetíveis ao ouvido humano e os filtros são mais estreitos. É por esse motivo que quando possuímos as energias do bando de filtros tomamos o logaritmo de cada uma (tudo devido à limitação do ouvido humano).

2.4 Transformada discreta do cosseno (DCT)

Como existe sobreposição das energias do banco de filtros então existe uma grande correlação entre elas, para desfazer esse facto usamos a transformada discreta do cosseno.

3 Projeto

Usei os áudios dos ficheiros "1_Atraining_normal, set_a" disponíveis na página do desafio http://www.peterjbentley.com/heartchallenge/ mas, para poder treinar o modelo, tive de criar um conjunto de ficheiros (amostras) na pasta "1_Splitedaudios" a partir dos originais onde os classificava eu mesma (S1, S2, ruído_1 e ruído_2) e usei para treinar o modelo. Neste modelo, de forma a possuir dados mais aceitáveis, juntei S1 e S2 e denominei só como S1 e o mesmo para os dois tipos de ruído.

Carregaram-se as seguintes dependências:

```
import IPython.display as ipd
import numpy as np
import pandas as pd
import librosa
import librosa.display
import matplotlib.pyplot as plt
from scipy.io import wavfile as wav
import os
from sklearn import metrics
from sklearn.metrics import $
    confusion_matrix, $
    ConfusionMatrixDisplay
from sklearn.preprocessing import $
    LabelEncoder
from sklearn.model_selection import $
    train_test_split
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, $
    Dropout, Activation
from tensorflow.keras.optimizers $
    import Adam
from tensorflow.keras.utils import $
    to_categorical

# To play audio file:
import IPython.display as ipd
```

Para conseguir observar o gráfico de uma onda sonora temos o seguinte código:

```
name = 'Wave_2.wav'
data, sample_rate = librosa.load(name$
    )
librosa_audio_data,$
    librosa_sample_rate=librosa.load($
    name)
plt.figure(figsize=(12, 4))
plt.plot(librosa_audio_data)
plt.show()
```

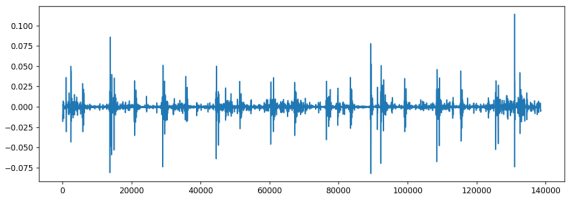

Figure 2. Exemplo de uma fórma de onda de um ficheiro audio

De forma a extrair todos os ficheiros áudio para treinar o modelo usámos o seguinte código com o auxílio de um ficheiro metadados externo:

```
for index, row in df.iterrows():
  file_name = df.loc[index]['fname']
  class_label = df.loc[index]['sound'$
      ]
  if 'noise' in class_label:
    class_label = 'noise'
    i+=1
  if (exists(file_name)):
    file_size = os.path.getsize($
        file_name)
    if (file_size > 44):
            data = extract_features($
                file_name)
            sounds.append([data, $
                class_label])
            cont += 1
n_mfccs = len(data)
featuresdf = pd.DataFrame(sounds, $
    columns=['soundtype','class_label'$
    ])
```

Tem de se ter em conta que o Librosa irá converter a frequência de amostragem (sample rate) para 22050 Hz automaticamente e irá normalizar os máximos e os mínimos do áudio para entre -1 e 1. Librosa converte também um sinal mono (duas fontes de som) para stereo (uma fonte). Necessita-se agora de retirar os MFCCs de cada áudio, deste modo criou-se a seguinte função:

```
def extract_features(file_name):
  audio, sample_rate = librosa.load($
      file_name)
  audio = shannonEnergy(audio)
  mfccs = librosa.feature.mfcc(y=$
      audio, sr=sample_rate, n_mfcc$
      =20, n_fft=512, hop_length=128)
```

```
    mfccs_processed = np.mean(mfccs.T,$
        axis=0)
    mfccs_out = mfccs_processed
    return mfccs_out
```

Primeiro convertemos os MFCCs em matrizes numpy e dividimos os nossos dados para módulos de treino ou de teste:

```
keras.backend.clear_session()
X = np.array(featuresdf.soundtype.$
    tolist())
y = np.array(featuresdf.class_label.$
    tolist())
# Encode the classification labels
le = LabelEncoder()
yy = to_categorical(le.fit_transform($
    y))
# Trainning and test sets
x_train, x_test, y_train, y_test = $
    train_test_split(X, yy, test_size$
    =0.2, random_state = 127)
```

Com estes dados e com as classes de cada ficheiro conseguimos agora construir e treinar o nosso modelo. Definimos e compilamos uma arquitetura de rede neural simples:

```
num_labels = yy.shape[1]
filter_size = 2
def build_model_graph(input_shape$
    =(40,)):
    model = Sequential()
    model.add(Dense(512))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(512))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(512))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_labels))
    model.add(Activation('softmax'))

    # Compile the model
    model.compile(loss='$
        categorical_crossentropy', $
        metrics=['accuracy'], optimizer=$
        'adam')
    return model

model = build_model_graph()
model(x_train)
model.summary()# Calculate pre-$
    training accuracy
score = model.evaluate(x_test, y_test$
    , verbose=0)
accuracy = 100*score[1]
print("Pre-training accuracy: %.4f%%"$
    % accuracy)
```

Vendo o resumo do modelo e observando a precisão do pré-treino obtemos os seguintes dados:

```
=================================================================
dense (Dense)                (324, 512)                10752
activation (Activation)      (324, 512)                0
dropout (Dropout)            (324, 512)                0
dense_1 (Dense)              (324, 512)                262656
activation_1 (Activation)    (324, 512)                0
dropout_1 (Dropout)          (324, 512)                0
dense_2 (Dense)              (324, 512)                262656
activation_2 (Activation)    (324, 512)                0
dropout_2 (Dropout)          (324, 512)                0
dense_3 (Dense)              (324, 2)                  1026
activation_3 (Activation)    (324, 2)                  0
=================================================================
Total params: 537,090
Trainable params: 537,090
Non-trainable params: 0
```

Figure 3. Dados do pré-treino

Depois de o modelo estar treinado, podemos avaliá-lo quanto à perda do teste e do treino, surgindo os seguintes valores a partir deste código:

```
score = model.evaluate(x_train, $
    y_train, verbose=0)
print("Training Accuracy: {0:.2\%\}".$
    format(score[1]))
score = model.evaluate(x_test, y_test$
    , verbose=0)
print("Testing Accuracy: {0:.2\%\}".$
    format(score[1]))
```

```
Training Accuracy: 99.38%
Testing Accuracy: 95.06%
```

Figure 4. Valores obtidos da precisão do teste e do treino

É também importante observar os valores referentes à perda, precisão (valores do teste), "val_loss" e "val_accuracy" (valores do treino).

```
Epoch 698/700
21/21 [==============================] - 0s 6ms/step - loss: 0.0097 - accuracy: 0.9969 - val_loss: 0.2693 - val_accuracy: 0.9383
Epoch 699/700
21/21 [==============================] - 0s 6ms/step - loss: 0.0136 - accuracy: 0.9969 - val_loss: 0.2566 - val_accuracy: 0.9383
Epoch 700/700
21/21 [==============================] - 0s 6ms/step - loss: 0.0303 - accuracy: 0.9969 - val_loss: 0.2369 - val_accuracy: 0.9383
```

Figure 5. Valores de perta e precisão do teste e do treino.

Observamos que o valor de "loss" na imagem acima é inferior ao "val_loss" e que a "accuracy" é superior à "val_accuracy" sensivelmente até à epoca 100, que corresponde ao comportamento desejado. Conseguimos observar a sua variação ao longo das épocas com os seguintes gráficos:

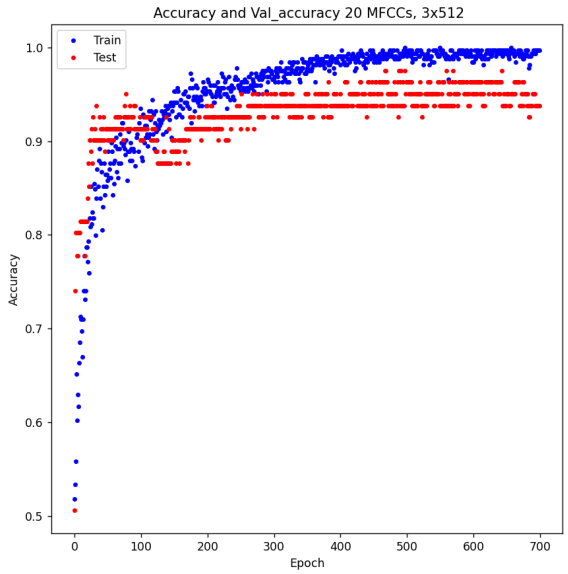

Figure 6. Gráfico da "accuracy" e "val_accuracy" ao longo das épocas

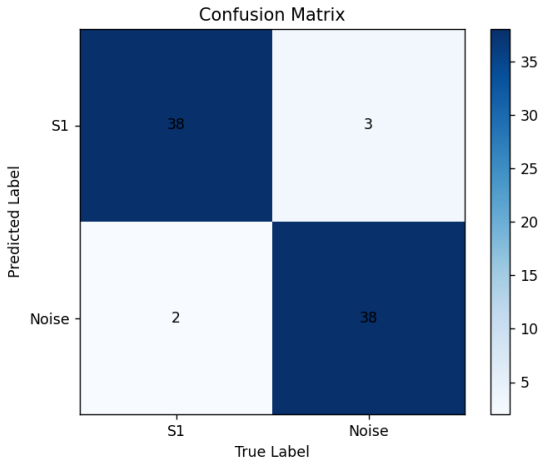

Figure 8. Matriz de confusão

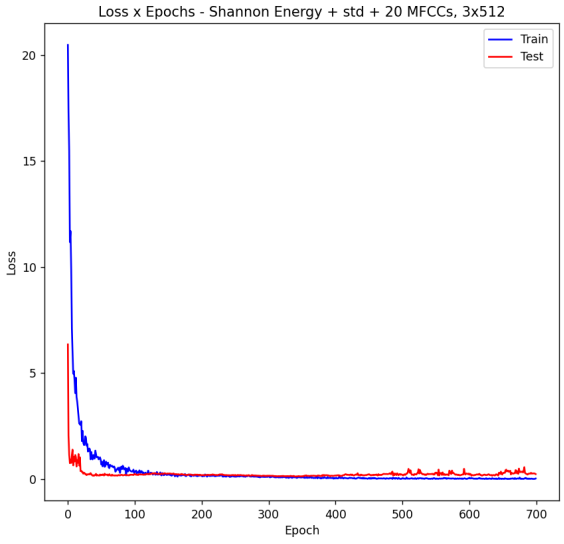

Figure 7. Gráfico da variação do "loss" e "val_loss" ao longo das épocas

Surgiu como desafio a criação de uma matriz de confusão:

Onde conseguimos observar que o modelo em 81 tentativas houve 38 ruídos e 38 S1 detetados corretamente e só 3 é que o modelo detetou como S1 e na realidade era ruído e 2 detedados como ruído quando na verdade eram S1. É de relembrar novamente que neste modelo juntei S1 e S2 e o mesmo para os dois ruídos. Quanto à inferência através do código abaixo conseguimos obter o gráfico seguinte:

```
fig ,ay1 = plt.subplots()
ay1.plot((librosa_audio_data$
    *(1/0.175))+0.5)
ay2 = ay1.twiny()
ay2.plot(timeDomain, predictions$
    [:,0], label='S1')
ay2.plot(timeDomain, predictions$
    [:,1], label='Noise')
plt.xlabel("Tempo - Segundos")
plt.ylabel("Probabilidade (\%)")
plt.legend()
plt.show()
```

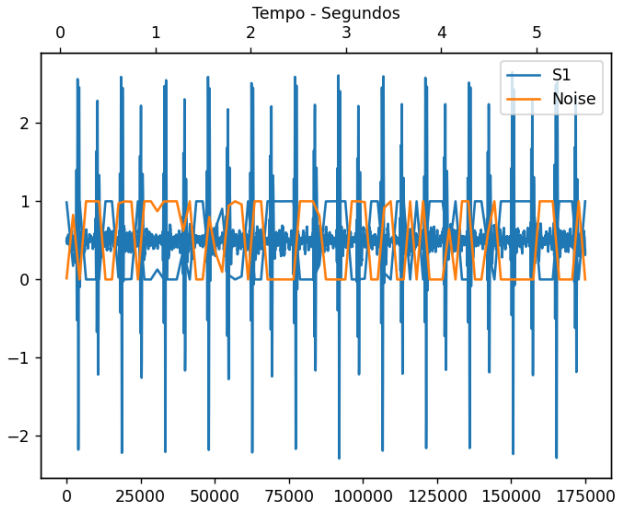

Figure 9. Gráfico das previsões e da onda original

É de notar que o ficheiro audio colocado para ser aplicado o modelo não foi usado para treinar a rede neuronal.

Através da média movel conseguimos então encontrar os picos que identificam o início de cada batimento:

```
#Copy of the predictions vector
mvAvg = np.copy(predictions,"K")

#length to stop the for lop, can sum $
    initial and final indexes
stopSize = np.size(predictions,0)
#lenght of the sum
sizeAvg = 1
#starting index
indexCount = sizeAvg

for item in predictions:
  if indexCount < stopSize - sizeAvg:
    mvAvg[indexCount][0] = movingAvg($
        predictions, indexCount, $
        sizeAvg, 0)
    mvAvg[indexCount][1] = movingAvg($
        predictions, indexCount, $
        sizeAvg, 1)
  indexCount += 1

# Max values
maxAvgOne = findPickInt(mvAvg, 0.2, "$
    S1")
maxAvgTwo = findPickInt(mvAvg, 0.4, "$
    Noise")
picAvgOne = maxS(maxAvgOne)
picAvgTwo = maxS(maxAvgTwo)
print(heartBeat(picAvgOne, timeDomain$
    , "S1"))
print(heartBeat(picAvgTwo, timeDomain$
    , "Noise"))

# Plot
plt.figure(figsize=(15,5))
plt.plot(mvAvg[:,0], label='S1mv')
plt.plot(mvAvg[:,1], label='Noisemv')
plt.scatter(picAvgOne[:,0],picAvgOne$
    [:,1], c='black')
plt.scatter(picAvgTwo[:,0],picAvgTwo$
    [:,1], c='red')
plt.show()
```

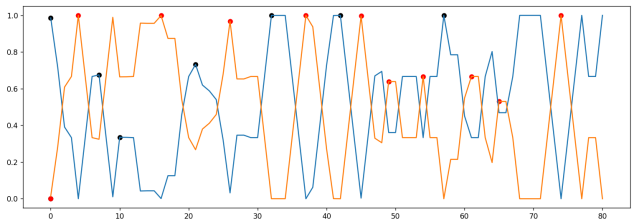

Figure 10. Código da média móvel e gráfico que regista o início de cada batimento

Depoisde calcular a diferença de tempo entre cada pico (identifica o início de cada batimento) conseguimos então determinar o ritmo cardíaco. É importante não esquecer que se juntou S1 com S2 e os dois tipos de ruído.

É de notar tambem que, os valores adquiridos automáticamente são médias. Na prática, o que se faz é: 1/(soma dos desvios/quantidade de desvios) = quantidade de desvios/soma dos desvios e este valor encontra-se em Hz. De forma a mudar para batimentos por minuto é simplesmente multiplicar por um fator de 60.

```
Batimentos: 1.34Hz ou 80.18 batimentos por minuto.
Calculado com S1.
Batimentos: 1.79Hz ou 107.27 batimentos por minuto.
Calculado com Noise
```

Figure 11. Batimentos cardíacos do paciente calculados com S1 e ruído

Através dos cálculos conseguimos observar que os batimentos cardíacos se encontram dentro do limite (60 a 100 batimentos por minuto). Comparando estes valores com os que se determinam ouvindo o ficheiro de som e contando manualmente os batimentos, que para S1 irá dar 90,23 batimentos/min e com o ruído é cerca de 93,98 batimentos/min, concluí-se que as incertezas são da ordem de 10,05% no cálculo com S1 e 13,29% no cálculo com o ruído. Esta discrepância de valores deve-se ao de apesar de treinado, o modelo não conseguir separar a 100% o S1 do ruído. As diferenças de valores entre S1 e ruído é devido à sua frequência no áudio. Por exemplo, calcular os batimentos com S1 é simplesmente contar os picos presentes no gráfico. Contudo, para o ruído este pode existir tanto antes como depois de cada pico podendo ser mais frequente ou igual conforme o ficheiro em análise.

Outro desafio que surgiu foi o de tentar otimizar a rede para melhorar os resultados. O que fiz inicialmente foi alterar o optimizador “Adam” que se encontra relacionado com a taxa de aprendizagem. Por defeito este possui um valor de cerca 0.001 e tinha-o alterado para 0,0003.

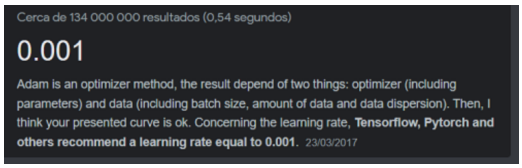

Figure 12. Definição do optimizador Adam

Contudo, a alteração do Adam não funciona em todos os casos.

Em áudios que ajudaram no treino do modelo observou-se descidas de cerca de 0.31% no Train Accuracy e aumento do Test Accuracy de 1,23%. Esta alteração foi feita for desconfiança de subtreinamento nas primeiras camadas. No entanto, para ficheiros que não foram usados para o treino os valores em geral pioram.

Algo que se poderia melhorar, acima de tudo, seria a quantidade e a qualidade das amostras, visto que, 405 ficheiros são, de todo, insuficientes para obter dados precisos. O estudo mais aprofundado da densidade das camadas e a sua quantidade seriam igualmente uma mais valia para melhoria dos resultados.

References

http://www.peterjbentley.com/heartchallenge/

https://www.youtube.com/watch?v=ZqpSb5p1xQo&ab_channel=MedallionDataScience

https://www.youtube.com/watch?v=aOv9c60KrJQ&ab_channel=HowTo

https://www.youtube.com/watch?v=h6X2THHnQ4s&ab_channel=ArdianUmam

https://www.youtube.com/watch?v=w8yWXqWQYmU&t=77s&ab_channel=SamsonZhang

https://www.youtube.com/watch?v=637eGwB0hfk&t=1427s&ab_channel=ValerioVelardo-TheSoundofAI

https://www.youtube.com/watch?v=udbA7u1zYfc&t=516s&ab_channel=LowByteProductions

https://www.youtube.com/watch?v=qmqCYC-MBQo&ab_channel=Siddhardhan

https://www.youtube.com/watch?v=iIkJrwVUl1c&t=3038s&ab_channel=CodeWithHarry

https://www.youtube.com/watch?v=vP06aMoz4v8&t=411s&ab_channel=StatQuestwithJoshStarmer

https://www.youtube.com/watch?v=ZLIPkmmDJAc&t=3437s&ab_channel=NicholasRenotte

https://www.youtube.com/watch?v=vmEHCJofslg&t=910s&ab_channel=KeithGalli

https://www.youtube.com/watch?v=s1XiCh-mGCA&ab_channel=PrettyPrinted

https://www.youtube.com/watch?v=JlwobCSFRBU&ab_channel=JimFikes

https://www.youtube.com/watch?v=Isa-qo_vXnw&t=103s&ab_channel=SoundDesign%26Tools

https://www.youtube.com/watch?v=uTFU7qThylE&t=308s&ab_channel=KrishNaik

https://www.youtube.com/watch?v=6_2hzRopPbQ&t=187s&ab_channel=NicholasRenotte

https://www.youtube.com/watch?v=QPDsEtUK_D4&t=2203s&ab_channel=Simplilearn

https://www.youtube.com/watch?v=bzZzv7zFRZM&t=737s&ab_channel=Prodramp

https://www.youtube.com/watch?v=wQ8BIBpya2k&t=1073s&ab_channel=sentdex

https://www.youtube.com/watch?v=MWYRGLKMzAQ&ab_channel=Kite

https://www.youtube.com/watch?v=irGGCHTLTkM&t=513s&ab_channel=Topictrick

https://www.youtube.com/watch?v=q5uM4VKywbA&t=125s&ab_channel=CoreySchafer

https://www.youtube.com/watch?v=637eGwB0hfk&t=747s&ab_channel=ValerioVelardo-TheSoundofAI

https://www.youtube.com/watch?v=bnHHVo3j124&t=115s&ab_channel=ValerioVelardo-TheSoundofAI

https://towardsdatascience.com/how-to-apply-machine-learning-and-deep-learning-methods-to-audio-analysis-615e286fcbbc

https://www.youtube.com/watch?v=O_oFF93D2pw&ab_channel=JoeCrow-TheAudioPro

https://www.youtube.com/watch?v=dBwr2GZCmQM&t=22s&ab_channel=AlilaMedicalMedia