Audio
Classification

Aprendizagem Computacional

Realizado por:

João Silva up202107600
Leonardo Regadas up202108144
Diogo Miranda up202106743

Índice

1. Introdução

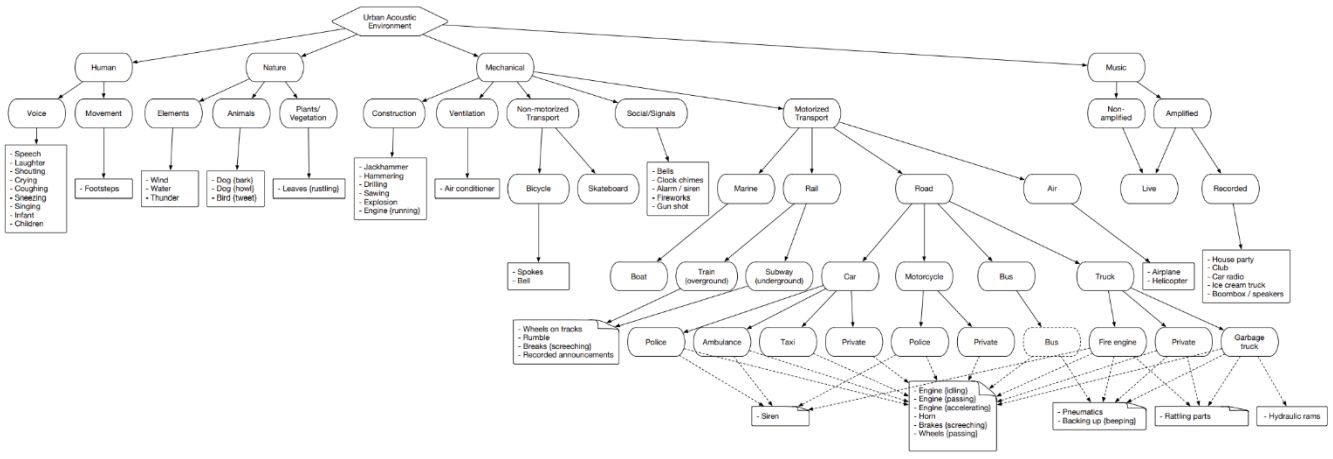
Contexto do problema

No nosso dia a dia encontramo-nos rodeados de vários sons, quer naturais, provocados pelo ser humano, ou ainda de várias outras formas.

Ainda assim, sem muito esforço, conseguimos distingui-los, mas como conseguimos fazer esta mesma tarefa com **redes neuronais** (deep learning)?

Assim, o objetivo deste trabalho é justamente desenvolver **deep learning classifiers** para classificar diferentes sons do dataset "urbansound8k", onde estes podem ser atribuídos a qualquer uma das seguintes classes:

- ar condicionado;
- buzina de um carro;
- crianças a brincar;
- cão a ladrar;
- som de perfuração;
- som de motor;
- som de tiro;
- jackhammer;
- sirene;
- música de rua.

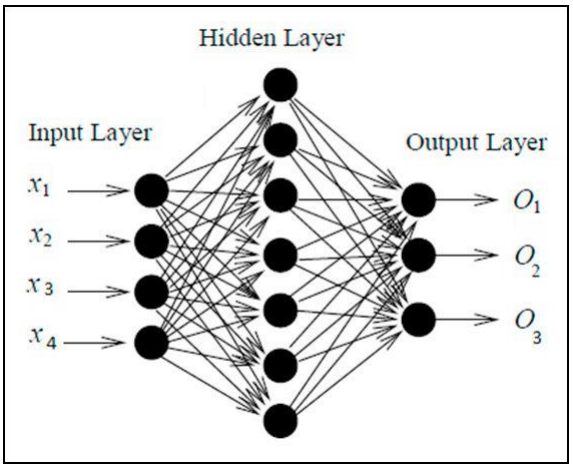


Descrição do MLP e CNN

Para resolver este problema proposto, optamos por usar os seguintes deep learning classifiers:

- classifier baseado num "**multilayer perceptron**";
- classifier baseado numa "**convolutional neural network**";

O **Multilayer Perceptron (MLP)** é um tipo de rede neuronal que, tal como no nome é referido, é estruturado por várias camadas, onde uma delas corresponde a uma **input layer**, seguida por uma ou mais camadas ocultas (**hidden layers**) com uma função de ativação associada, responsáveis pela maior parte do processamento dos dados, e acabando por uma **output layer**, que, neste tipo de problemas (classificação), é constituída por um número de neurónios igual ao número de classes do problema.



Já as **redes neuronais convolucionais (CNN)** são redes que recorrem a um processamento de dados em forma de imagens. São constituídas por **camadas convolucionais** que, com um conjunto de filtros, processam as imagens de entrada para produzir certas características que passam para camadas de **ativação** e, depois, para camadas de **pooling**, necessárias para reduzirem a quantidade excessiva de parâmetros da rede. Por fim, as informações prosseguem para camadas **totalmente conectadas**, necessárias para classificar as características aprendidas nas camadas anteriores em categorias específicas. Tal como o MLP, a rede acaba numa **output layer**.

2. Estrutura do trabalho

Para desenvolver este trabalho, começamos por dividi-lo em **4** etapas principais:

- **Análise** e **visualização** da informação;
- **Pré-processamento** e **preparação** dos dados;
- **Construção**, **treino** e **resultados** do modelo para **MLP**;
- **Construção**, **treino** e **resultados** do modelo para **CNN**;

Começamos por analisar o dataset fornecido, assim como as pastas com os áudios e os áudios em si (**Notebook 1** "data_analysis").

Pegamos nos dados e aplicamos um pré-processamento de forma a termos features extraídas para cada um dos modelos (**Notebook 2** "data_preprocessing")

Com essas features, construímos um modelo para cada classifier escolhido, treinamos e retiramos algumas estatísticas dos mesmos (**Notebook 3** "mlp_implementation" e **Notebook 4** "cnn_implementation").

Além disso, guardamos em novos **ficheiros .csv** as features extraídas e os resultados da accuracy para cada classe.

Todos os notebooks e ficheiros encontram-se disponíveis no .zip enviado e no seguinte github:

zJohnnyPT/DeepLearningClassifiers (github.com)
(Ctrl + clique esquerdo para abrir)

3. Implementação

Pré-processamento

Primeiramente, queremos que os áudios estejam todos **uniformizados** em termos de duração e queremos **normalizá-los** de forma a que nos ajude no tratamento dos mesmos, uma vez que estes podem ter características com escalas bastante diferentes que acabariam por afetar o processo de aprendizagem.

Assim, começamos por:

- transformar todos os áudios com diferentes durações em áudios com **4 segundos de duração**;
- normalizar as características dos áudios para uma escala de **[-1, 1]**.

Agora, com os áudios devidamente tratados, prosseguimos para a **extração de features** dos mesmos. Para tal, usámos dois métodos:

- **mel-frequency cepstral coefficient**s (MFCC);
- **mel-scaled** spectrograms;

Através do MFCC, obtemos features na forma de um **array**, que servirá como input para os **multilayer perceptrons**.

Com os **spectrograms** obtidos no mel-scaled, reunimos as características e os requisitos necessários para conseguir construir um modelo em **CNN**.

Tal como já foi referido, ambas as features foram guardadas em dois ficheiros excel:

- df_features_mels.csv;
- df_features_mfcc.csv.

Além disso, e uma vez que nos facilitou na hora de treinar os modelos, guardamos também as features em **ficheiros .npy**, ficando os datasets criados apenas para uma melhor e mais prática visualização das features.

Construção dos modelos

Em ambos os modelos, começamos por fornecer o **número de classes** do problema (10) e o **formato de input** que as features têm após extraídas.

Para o modelo do MLP, criamos **4 hidden layers**, começando com 1024 filtros na primeira camada e a cada camada que aumentava o número de filtros diminuía para metade. Para todas estas camadas, foi definida a **função de ativação "relu"**. Após cada camada, é utilizado o **BatchNormalization()** e o **Dropout()** com valor de **0.3** em todos os casos, o que indica que existe 30% de chance de remover alguma feature. Este processo permite **evitar o overfitting**. Após estas 4 camadas, é moldado o formato do output com auxílio do **Flatten()** e termina por criar o **output layer** com um número de neurónios igual ao número de classes e recorrendo à função **softmax** como função de ativação.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_107 (Dense)           (None, 40, 1024)          178176

 batch_normalization_80 (Ba  (None, 40, 1024)          4096
 tchNormalization)

 dropout_80 (Dropout)        (None, 40, 1024)          0

 dense_108 (Dense)           (None, 40, 512)           524800

 batch_normalization_81 (Ba  (None, 40, 512)           2048
 tchNormalization)

 dropout_81 (Dropout)        (None, 40, 512)           0

 dense_109 (Dense)           (None, 40, 256)           131328

 batch_normalization_82 (Ba  (None, 40, 256)           1024
 tchNormalization)

 dropout_82 (Dropout)        (None, 40, 256)           0

 dense_110 (Dense)           (None, 40, 128)           32896

 batch_normalization_83 (Ba  (None, 40, 128)           512
 tchNormalization)

 dropout_83 (Dropout)        (None, 40, 128)           0

 flatten_15 (Flatten)        (None, 5120)              0

 dense_111 (Dense)           (None, 10)                51210

=================================================================
Total params: 926090 (3.53 MB)
Trainable params: 922250 (3.52 MB)
Non-trainable params: 3840 (15.00 KB)
```

Construção dos modelos

Na implementação do CNN, inicialmente são adicionados ao modelo **2 convolutional layers** com 32 filtros e a **função de ativação "relu"**. Para reduzir as dimensões do input, já que se tratavam de features 2D, é utilizado o **MaxPooling2D** com uma área de **2x2** e o **SpatialDropout2D()** para prevenir o overfitting. A diferença entre o dropout utilizado no MLP e a função de dropout utilizada no CNN é que aqui queremos remover conjuntos completos 2D de features e não apenas alguns dos valores do conjunto. Com isto, decidimos reduzir a chance de dropout para 10% na primeira vez e para 20% na segunda. Por fim, queremos converter estas features 2D para um array 1D para conseguir criar a camada de output. Para isto, foi utilizado o **Flatten()** e foi criada uma **camada totalmente conectada** antes da própria camada de output final.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 38, 172, 32)       320

 max_pooling2d_8 (MaxPoolin  (None, 19, 86, 32)        0
 g2D)

 spatial_dropout2d_8 (Spati  (None, 19, 86, 32)        0
 alDropout2D)

 conv2d_9 (Conv2D)           (None, 17, 84, 32)        9248

 max_pooling2d_9 (MaxPoolin  (None, 8, 42, 32)         0
 g2D)

 spatial_dropout2d_9 (Spati  (None, 8, 42, 32)         0
 alDropout2D)

 flatten_4 (Flatten)         (None, 10752)             0

 dense_8 (Dense)             (None, 128)               1376384

 dense_9 (Dense)             (None, 10)                1290

=================================================================
Total params: 1387242 (5.29 MB)
Trainable params: 1387242 (5.29 MB)
Non-trainable params: 0 (0.00 Byte)
```

Para ambos os modelos, usamos o **otimizer "Adam"** e como **loss** utilizamos o **"categorical_crossentropy"**.

Para os treinar, decidimos também utilizar um total de **20 epochs** para cada fold e um **batch_size de 32**. Consideramos estes valores ideais para um bom **tradeoff** entre a **accuracy** e o **tempo de treino** do modelo. Por outras palavras, acreditamos que estes hyperparameters conferiram um bom número total de parâmetros.

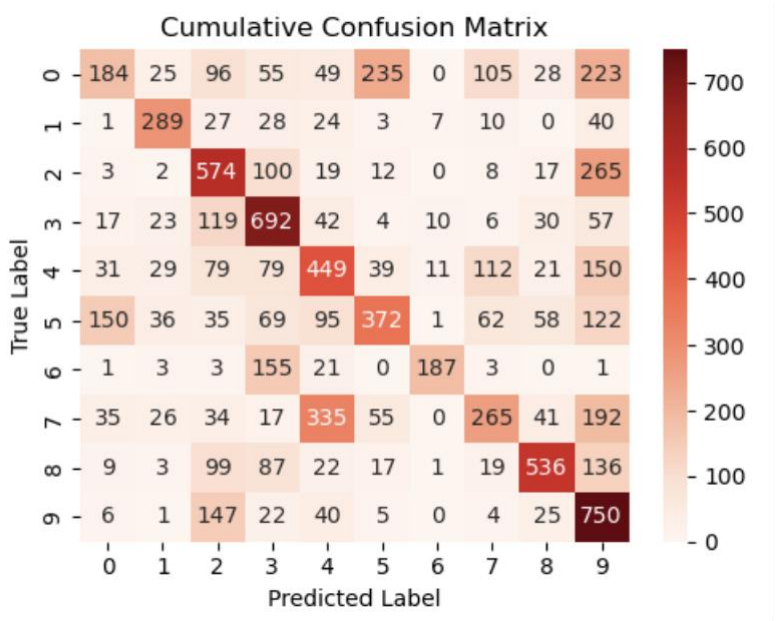
4. Resultados

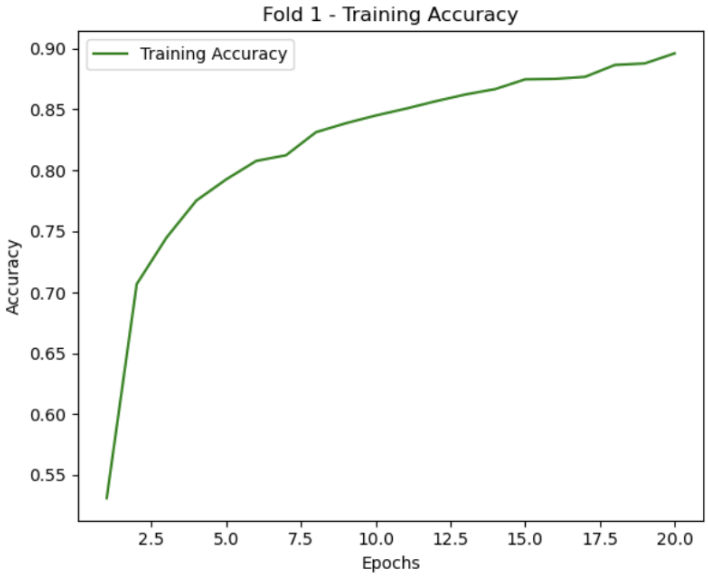
Para cada rede neuronal / modelo, conseguimos retirar as seguintes estatísticas:

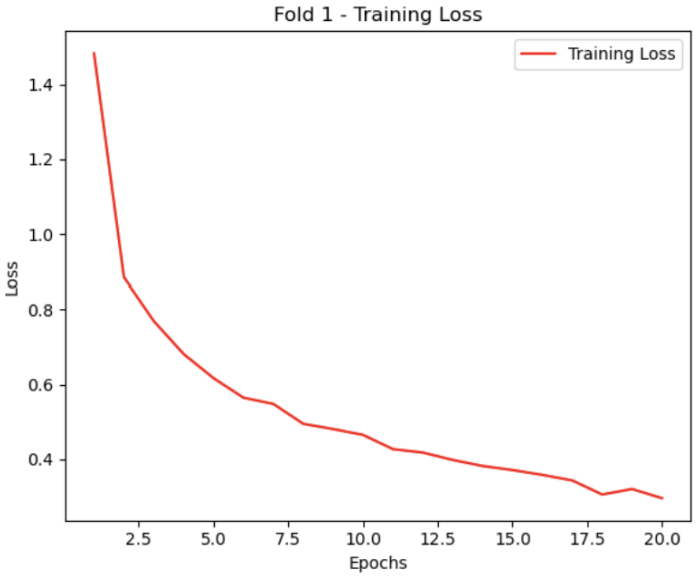
- **Matrizes de confusão** para cada folder;
- **Matriz de confusão comulativa**;
- **Gráficos de training loss** e **training accuracy** ao longo do num_epochs;
- **Accuracy** de cada **classe**;
- **Average accuracy** e **standard deviation** para training e para test.

Não vamos apresentar aqui todas as matrizes de confusão e todos o gráficos de training loss e training accuracy por termos estas estatísticas para cada fold. Para visualizar todas as estatísticas por completo, consultar o **notebook_3** e **notebook_4**.

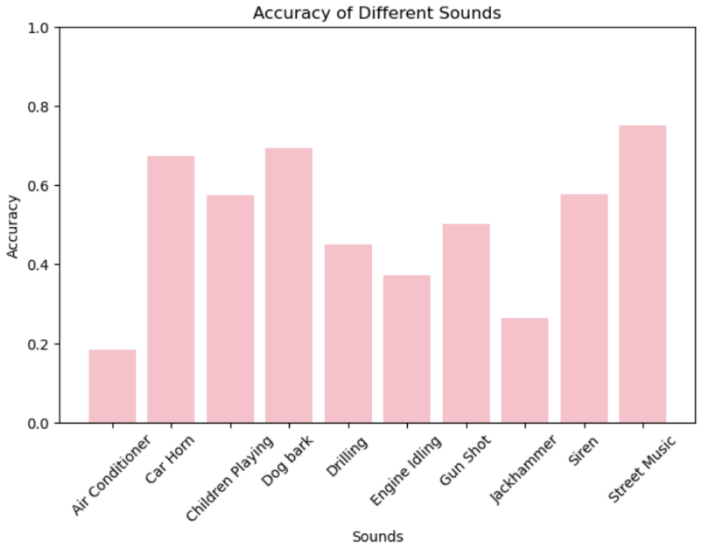
MLP Classifier







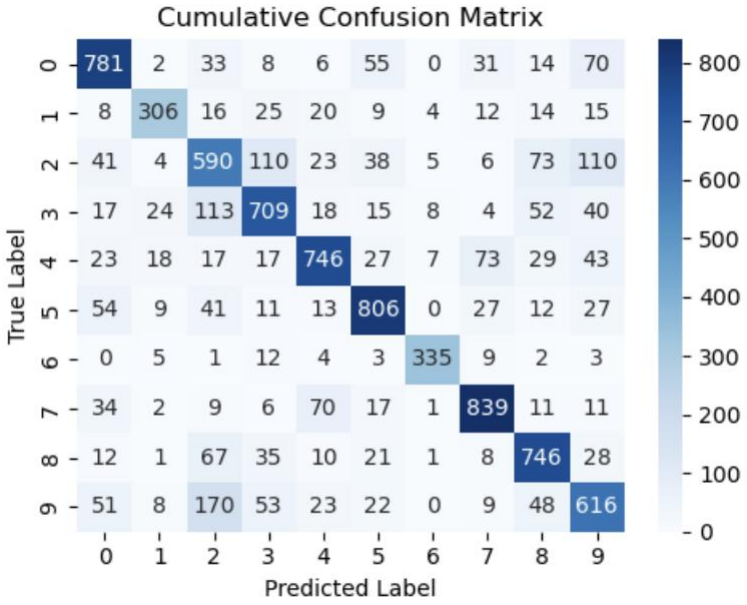
	CLASS	ACCURACY
0	Air Conditioner	0.184000
1	Car Horn	0.673660
2	Children Playing	0.574000
3	Dog bark	0.692000
4	Drilling	0.449000
5	Engine Idling	0.372000
6	Gun Shot	0.500000
7	Jackhammer	0.265000
8	Siren	0.576964
9	Street Music	0.750000

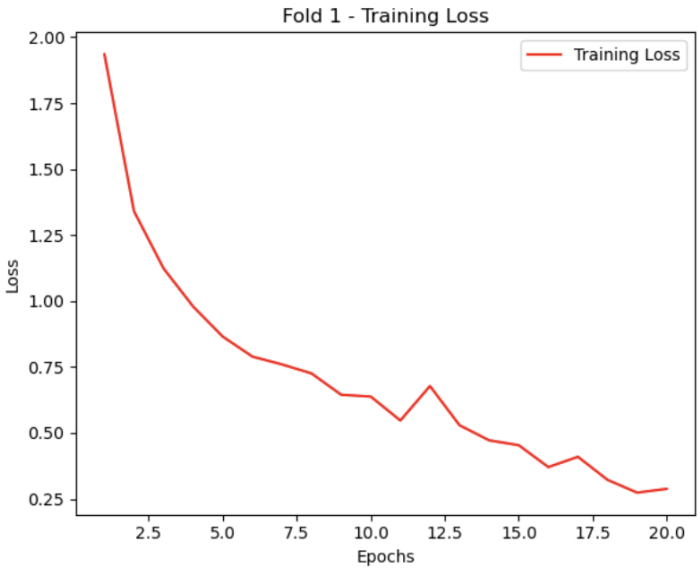


Average training accuracy: 0.8973125398159028
Standard training deviation: 0.005060631396380184
Average test accuracy: 0.49221255153458543
Matrix deviation: 143.23636968312204

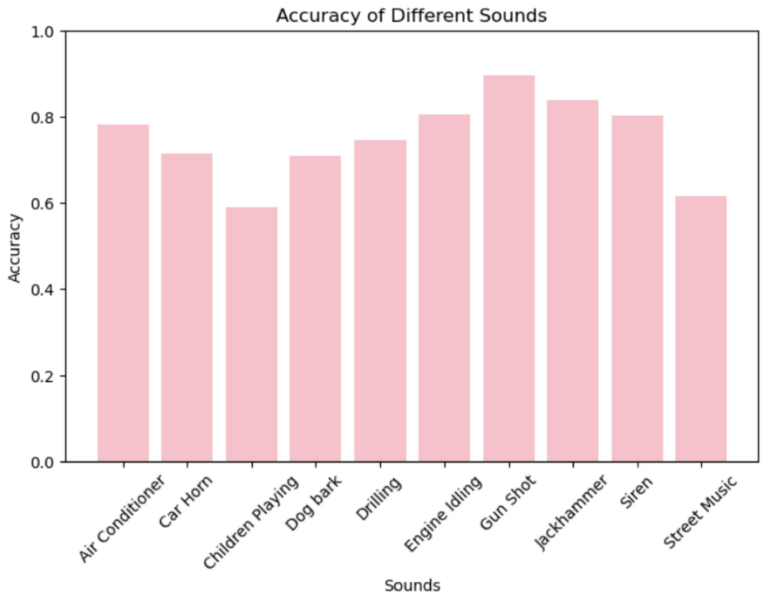
4. Resultados

CNN Classifier





	CLASS	ACCURACY
0	Air Conditioner	0.781000
1	Car Horn	0.713287
2	Children Playing	0.590000
3	Dog bark	0.709000
4	Drilling	0.746000
5	Engine Idling	0.806000
6	Gun Shot	0.895722
7	Jackhammer	0.839000
8	Siren	0.803014
9	Street Music	0.616000



Average training accuracy: 0.90302594304084 78
Standard training deviation: 0.011933663812720109
Average test accuracy: 0.7414109024278516
Matrix deviation: 197.05572206865756

5. Comentários finais e conclusões

Falando de accuracy, foi possível observar resultados bastante mais positivos no CNN em relação ao MLP. Acreditamos que o uso de espectrogramas neste problema tenha ajudado bastante a criar um modelo mais eficiente para a classificação dos diferentes sons.

No modelo MLP, notou-se uma grande dificuldade na classificação dos seguintes sons:

- ar condicionado;
- jackhammer.

Mais concretamente, o modelo teve dificuldades em diferenciar os seguintes conjuntos de sons:

- ar condicionado e som de motor;
- ar condicionado e música de rua;
- drilling e jackhammer;

Comparando os valores de accuracy e de teste, concluímos que os resultados não foram os melhores pois há uma grande discrepância entre ambos. Assim, podem existir problemas de overfitting neste modelo.

Já quanto ao CNN, os resultados foram bem mais positivos.

Entre as diferentes estatísticas, realçamos o bom accuracy das seguintes classes:

- gun shot;
- jackhammer.

Como podemos ver, no CNN o jackhammer foi um dos melhores classificados, o que é o oposto do MLP.

Concluindo, acreditamos que o CNN está bem construído, ao passo que o MLP precisaria de ser melhorado mais um pouco para reduzir o possível overfitting.

6. Bibliografia e Webgrafia

UrbanSound8K - Urban Sound Datasets (weebly.com)

salamon_urbansound_acmmm14.pdf (justinsalamon.com)

https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141

https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

https://towardsdatascience.com/cnns-for-audio-classification-6244954665ab

https://www.geeksforgeeks.org/plotting-various-sounds-on-graphs-using-python-and-matplotlib/

https://www.analyticsvidhya.com/blog/2021/06/visualizing-sounds-librosa/

https://www.researchgate.net/publication/347356900_Audio_Pre-Processing_For_Deep_Learning

https://towardsdatascience.com/urban-sound-classification-part-1-99137c6335f9