Real-Time Identification of Bird Species using Machine Learning:

Introduction:

Have you ever heard a bird's call and wondered which bird is that?
In this project we present an Audio classification method for bird species identification. Our approach is based on Deep Learning (CNN). The data collected is pre-processed by separating it into Audio signal and Noise signal.The audio signal is usually transformed into a time-spectral representation. It was then followed by Data augmentation which is then trained using Convolutional Neural Networks. Our final CNN model is made to work in real time.
(Check out the GitHub link for code pieces of each part)

Tech stack:

Our project is built using Python, libraries used are :

Librosa: for extracting Audio signal features,

Tenserflow-Keras: for building and testing CNNs

Tkinter: for GUI of the app

Step -1: Data Collection

Our first step includes data collection. Since a machine learning project requires huge amount of data, we collected sound data (bird calls) of four birds from Xenocanto repository(xeno-canto :: Sharing bird sounds from around the world) using the Python
Those birds are :

1. Corvus Corax (Common Crow)
2. Cucculus Carorus (Cuckoo)
3. Parus Major (Blue tit)
4. Passer Domestics (Sparrow)

For each of these bird species 300 recording samples of varied length and another 100 recordings (separately for testing purpose) are used. Due to these audio files being in mp3 format (a compressed file format) we converted them into wav files (an uncompressed audio format suitable for audio processing). Now we have one directory for each bird with all its recordings in '.wav' format.

Step-2:Feature Generation and Preprocessing:

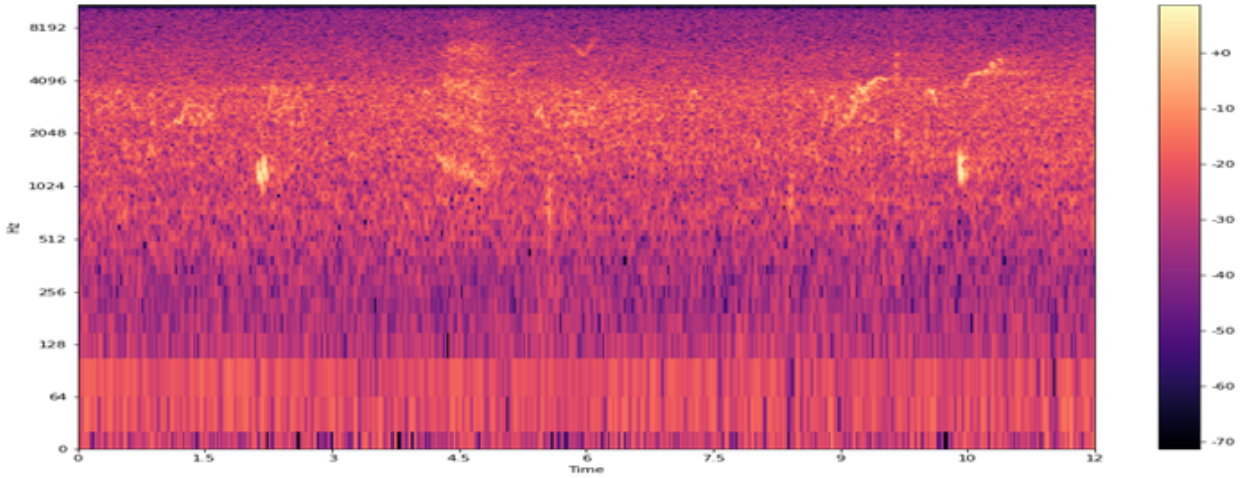
Pre-processing of audio files is essential for feature extraction.We use Spectrograms to extract the necessary features. But before that, we need to remove noise.

1)Signal and Noise Separation:

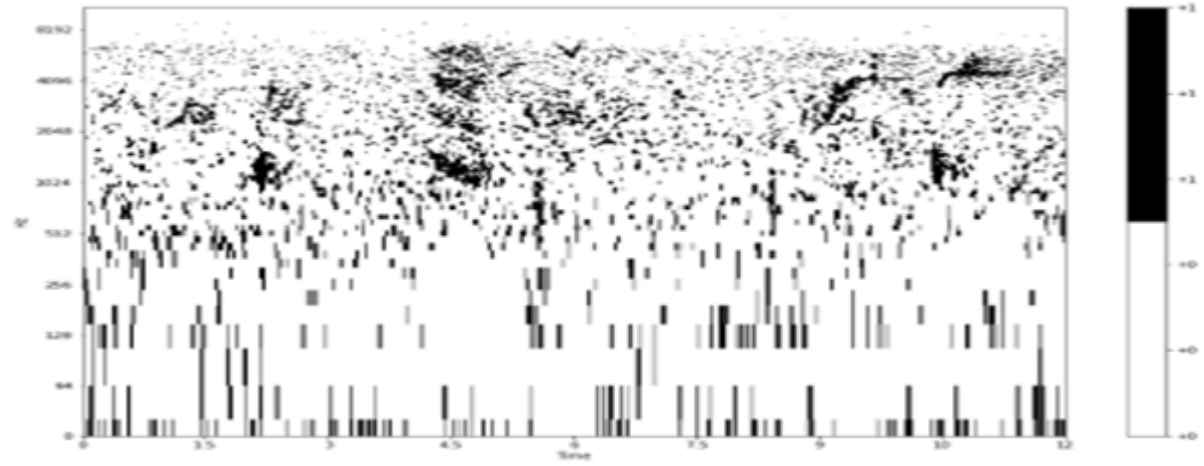
The .wav files are sampled at 22,050Hz. Here we have separated the Original Audio file into 3s Signal and 3s Noise Chunks using Sprengel Mask (the process is elaborated below).

Let's take a .wav recording of a bird and follow these steps.

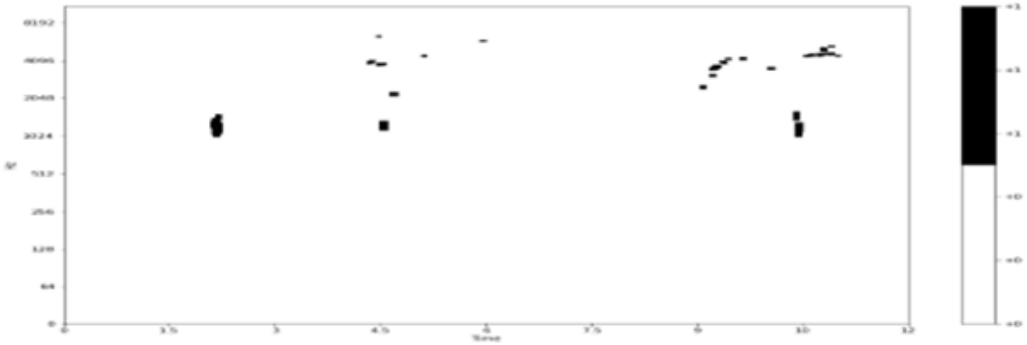
a)The Spectrogram of the Original Audio Signal.



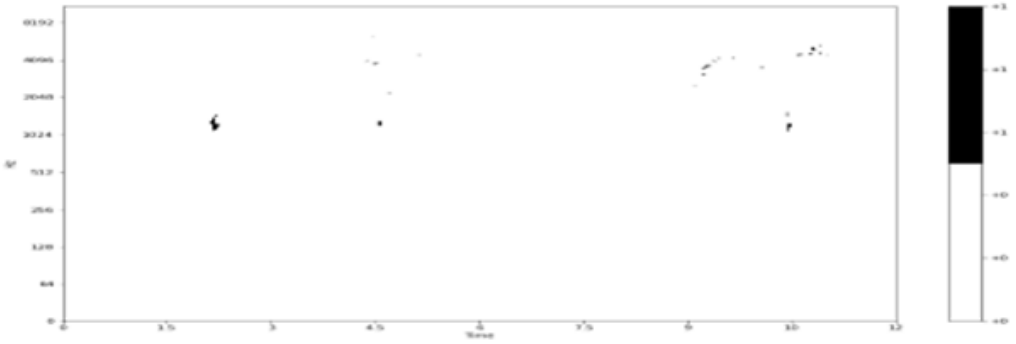
b)The Spectrogram is Normalised (Converted to a Binary Image) and Median Clipping is performed.



c)Binary Dilation is performed with the help of a 4x4 filter



d)Binary Erosion is done with the help of a 4x4 filter



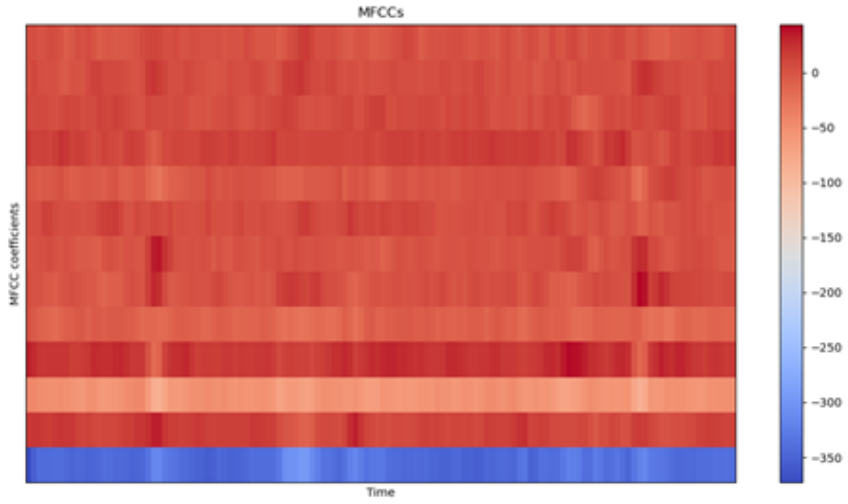
The selected pixels are shown as spots and only these parts are concatenated and considered as a signal, similar procedure in reverse gives noise. We neglect silent spots as they aren't useful. Finally we have 2 directories for each bird, one containing Signal and another containing noise, with 3s samples each.

2)MFCC

Now these audio recordings are transformed into MFCCs which is a more compact way of representing the recording.

Mel Frequency Cepstral Coefficients (MFCCs) have been a standard choice for extracting audio features used in speech recognition, and their success owes much to the amount of information they contain about the audio signal in such a compressed form.

Parameters to be considered, Hop Length=13; Hanning Window; Frame Size=512.



Ref links:

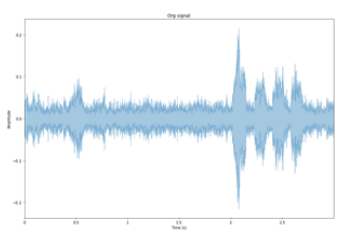
Code:jm_preprocess.py

Step-3 Data Augmentation

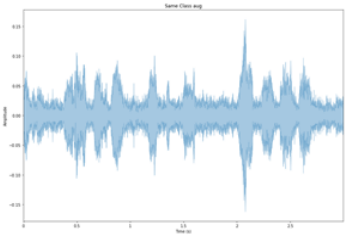
Data Augmentation is done to improve the number of Training samples and to make the Model more resilient to Noise.

1)Same Class and Noise Addition:

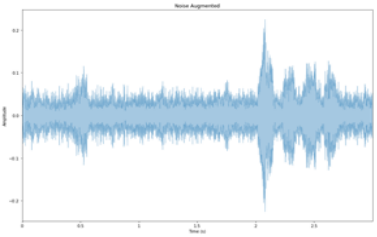
In order to improve the convergence rate of the neural network, the training samples are augmented by additively combining each sample with another same class sample. This lets the network see more relevant data at once. The samples are also additively combined with three random noise segments, to make the network more noise invariant, and therefore generalize better. Each sample shown to the neural network is thus a combination of two randomly chosen, same class signal segments x1 and x2, and three randomly chosen noise segments n1, n2, and n3 of the same length.



Audio sample



After same class augmentation



After Noise augmentation

Time Shift:

The time shift augmentation is done by splitting a spectrogram sample into two parts, along the time axis, and then placing the second part before the first. That is, a wrap-around shift in the time domain

Pitch Shift:
The pitch shift augmentation is done in a similar way, but in the frequency domain (vertically), and the shift is only around 5%. Larger shifts are said to not be beneficial

We use librosa functions to implement these techniques.MFCCs and their corresponding Class (Bird) are stored as pairs in a npz file,which is the input to the ML model.**(Train.npz,Valid.npz)**

Step-4: Convolutional Neural Networks:
Convolutional neural networks refer to a sub-category of neural networks that are specifically designed to process input images. Their architecture is composed of two main blocks:

The first block functions as a feature extractor. It performs template matching by applying convolution filtering operations. The first layer filters the image with several convolution kernels and returns “feature maps”, which are then normalized with an activation function, ReLU in our case.

In the second block, the input vectors are transformed to return a new vector to the output. This last vector contains as many elements as there are classes, with each element representing the probability that the image belongs to a particular class. These probabilities are calculated by the last layer which uses a softmax function as an activation function.

Load the training and validation sets we obtained in the last step as shown below

```
with np.load('/content/gdrive/My Drive/4_class_train.npz') as data:
    X_train = data['mfccs']
    y_train = data['labels']

with np.load('/content/gdrive/My Drive/4_class_valid.npz') as data:
    X_valid = data['mfccs']
    y_valid = data['labels']
```

We used the Keras API, a high-level Deep Learning API to build, train, and evaluate our CNN. Its documentation is available at **https://www.tensorflow.org/api_docs/python/tf/keras.**

After continuous experimentation, the CNN architecture we ended up with is shown below

```
def build_model(input_shape):
    model = keras.Sequential()

    model.add(keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=input_shape,padding='same'))
    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    model.add(keras.layers.Conv2D(32, (2, 2), activation='relu'))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(128, activation='relu'))
    model.add(keras.layers.Dense(5, activation='softmax'))

    return model
```

There are four convolution layers with ReLU activation in total with the last three layers being followed by pooling and batch normalization layers. Finally, the layers are flattened and sent to the fully connected layers for classification.

The convolutional layer -
It detects the presence of a set of features in the images received as input. This is done by convolution filtering.

The pooling layer -
It receives several feature maps and applies the pooling operation to each of them which consists of reducing the size of the images while preserving their important characteristics.

The ReLU Activation layer -
ReLU (Rectified Linear Units) refers to the non-linear function defined by ReLU(x)=max(0,x).

The Fully Connected layer -
It consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

Hyperparameters
Number of epochs -
It is the number of times the whole training data is shown to the network while training.
We chose an epoch count of 24 since anything higher than that was detrimental to the average accuracy of the model.

Batch size -

Mini batch size is the number of sub samples given to the network after which parameter update happens. We found that a batch size of 8 worked well for our use case.

Learning rate -
It defines how quickly a network updates its parameters. We found that a learning rate of 0.0001 gave the best results.

A summary of our CNN model is shown below

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 13, 517, 64)       640
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 515, 32)       18464
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 6, 258, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 6, 258, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 256, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 2, 128, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 2, 128, 32)        128
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 1, 127, 32)        4128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 64, 32)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 1, 64, 32)         128
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 128)               262272
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 645
=================================================================
Total params: 295,781
Trainable params: 295,589
Non-trainable params: 192
```

Step-5:Training and Prediction and GUI

The Training process of MFCCs with the created CNN model is completed with the specified hyperparameters. The model is saved in h5 format using model.save() function.

The Accuracy and Error plot of the trained model is plotted using the Matplotlib library.

Testing:
The above-obtained accuracy is on test samples obtained from the train_test_split function of scikitlearn, which means the test samples are augmented too. To get a stringent value, we need to test it with raw samples without augmentation.
We use the 100 samples we downloaded separately for this purpose. We create a **'test.npz'** file with the audio recordings of these samples along with their corresponding labels. This is loaded and converted into MFCCs and tested with the trained model. This Test accuracy is a more realistic measure of its efficiency.

A Confusion matrix can also be plotted to visualize the Classifications made.

```
[[ 0,  0,  0,  0,  0],
 [ 5, 81,  1, 13,  0],
 [ 8, 28, 52,  8,  4],
 [ 0, 14,  0, 75, 11],
 [ 3,  5,  0, 22, 70]]
```

The top row is filled with 0's as it corresponds to the Noise class and No particular sample is predicted as Noise as it doesn't have distinct features, it just helps to avoid misclassification of other classes. The Diagonal entries are larger depicting that there are more right classifications.

Step-6:Prediction and GUI:
Now to actually see how our model works in action. We now have a trained and tested model which is ready to identify the bird when a bird call recording is given.
Steps to Prediction:
1)Record audio of a bird call in real-time and save it as a '.wav' file.

2) Use librosa split function to do silence removal (so that we'll focus on the useful parts of the recording).

```
def silence_removal(wav1,db_val=10):
    sig_sec=librosa.effects.split(y=wav1, top_db=db_val)
```

'top_db' specifies the number of decibels below the ref value that it considers as signal and leaves the rest as silence. Concat the splits together and it is split into 3s chunks.

3) Each of these 3s chunks is transformed to MFCCs.

4)Now we have MFCCs, which can be given as input to the loaded model.

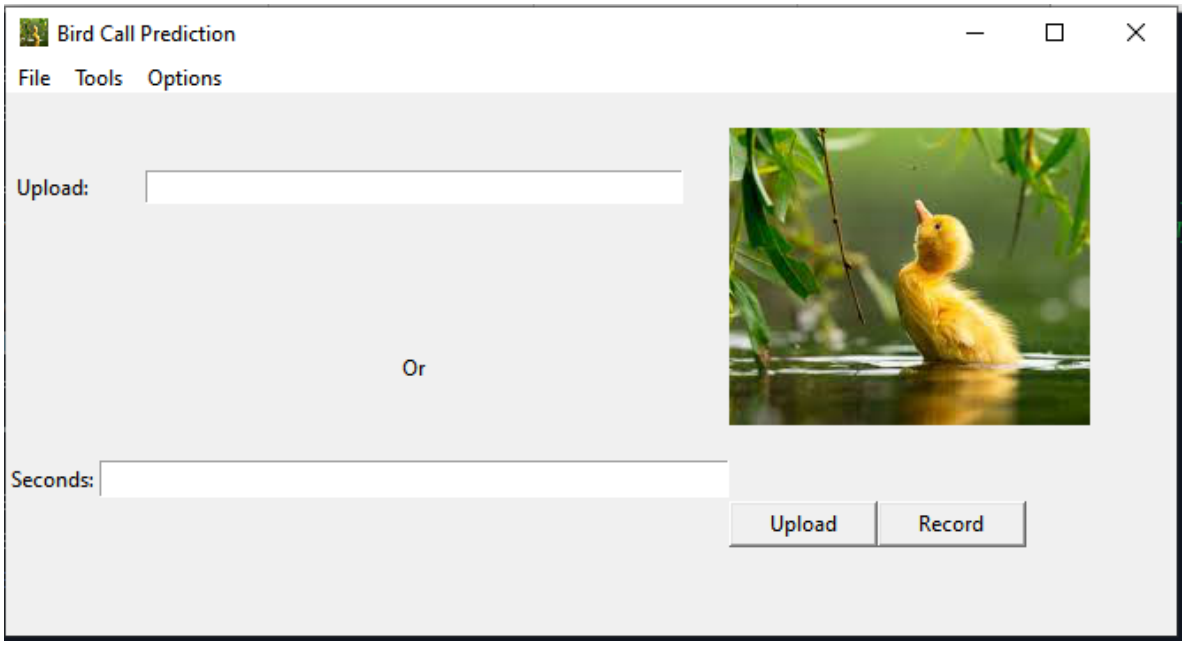
5) For each of the MFCCs a particular class is shown as a prediction and we need to take the class that was predicted the most number of times. And the Bird corresponding to this class is the Bird behind the initial call. Bird Identified.

```
[3 3 3 3 0 3 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
33
Total no.of clips predicted...33
Counter({'Parus major': 31, 'Noise': 2})
Predicted bird is...Parus major
```

Here,33 clips are predicted, among which 31 are predicted as Parus major and 2 are Noise(can be safely ignored) and so this seems to be the call of the Bird Parus major.

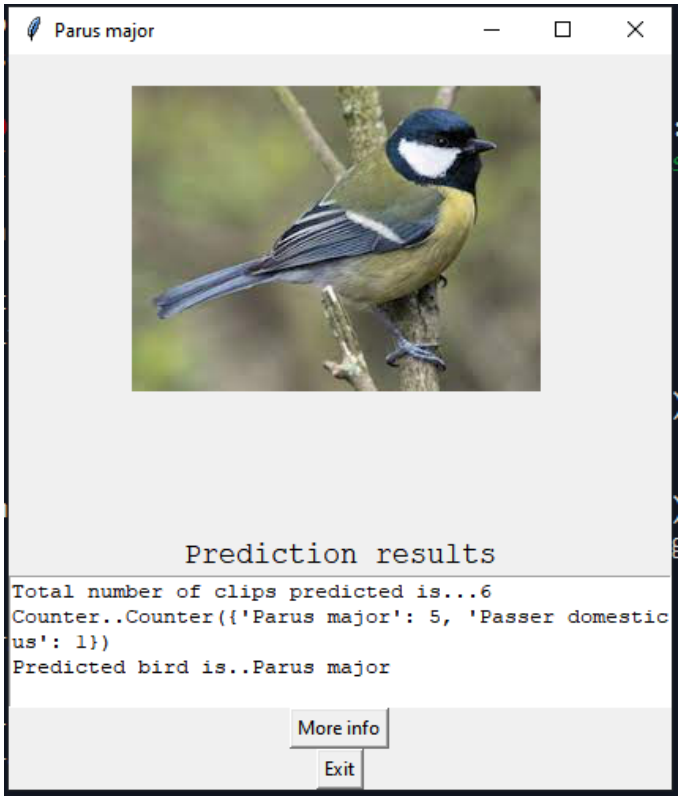
GUI using Tkinter:

Now, to make these steps easy and quick, a GUI interface is created which allows getting the results with the click of a button.



This GUI is created with the Tkinter Python library. (Code below) It has 2 options, either you can upload an already recorded audio or do real-time recording(get instant results!). Specify the number of seconds to record and press the record button to start recording and to upload using the upload button.

The MFCCs for each clip is shown and then the Predicted bird pops up along with the info of how many clips were classified as this bird. To know more about this bird press the More Info button which will take you to its Wiki page.



This GUI has options to change the model used so that we can upgrade with better models with time.

Conclusion:

Now, when you listen to a bird call, have this GUI ready to identify which Bird is that.
We prepared this with a basic setup and so it considers only 4 birds, but it can be easily extended to more birds, and with a large training dataset you can classify it with higher accuracy.
The More data we have, the better the model learns, and the better it classifies an unknown bird call.

Youtube video link:

https://www.youtube.com/watch?v=Gy_WHTzGzGg&t=338s

Github:
BharathKumarDP/Bird-Sounds-Classifier: A Deep learning model to identify and classify Bird Sounds using CNNs. (github.com)

Papers:

1.http://ceur-ws.org/Vol-1609/16090547.pdf -Sprengel
2.249467.pdf (chalmers.se)-John martinson (johnmartinsson/bird-species-classification: Using convolutional neural networks to build and train a bird species classifier on bird song data with corresponding species labels. (github.com)

References:
1.https://towardsdatascience.com/sound-event-classification-using-machine-learning-8768092beafc
2.https://github.com/musikalkemist/AudioSignalProcessingForML
3.https://www.youtube.com/channel/UCZPFjMe1uRSirmSpznqvJfQ
4.CNN-https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/
5.http://neuralnetworksanddeeplearning.com/