**IMPLEMENTATION**

**MODULES:**

* Dataset
* Importing the necessary libraries
* Exploratory Data Analysis of Audio data
* Imbalance Dataset check

## Data Preprocessing

* Splitting the dataset
* Audio Classification Model Creation
* Compile the Mode**l**
* Train the Model
* Check the Test Accuracy
* Saving the Trained Model

**MODULES DESCSRIPTION:**

**Dataset:**

In the first module, we developed the system to get the input dataset for the training and testing purpose.

Data set link: <https://www.kaggle.com/datasets/vinayshanbhag/bird-song-data-set>

The dataset consists of 5,422 Bird Sound Classification Using Deep Learning and can practice it on Kaggle itself.

**Importing the necessary libraries:**

The very important and great library that supports audio and music analysis is Librosa. Simply use the Pip command to install the library. It provides building blocks that are required to construct an information retrieval model from music. Another great library we will use is for deep learning modeling purposes is TensorFlow, and I hope everyone has already installed TensorFlow.

## Exploratory Data Analysis of Audio data

We have 5 different folders under the urban dataset folder. Before applying any preprocessing, we will try to understand how to load audio files and how to visualize them in form of the waveform. If you want to load the audio file and listen to it, then you can use the IPython library and directly give it an audio file path. We have taken the first audio file in the fold 1 folder that belongs to the dog bark category.

Now we will use Librosa to load audio data. So when we load any audio file with Librosa, it gives us 2 things. One is sample rate, and the other is a two-dimensional array. Let us load the above audio file with Librosa and plot the waveform using Librosa.

Sample rate – It represents how many samples are recorded per second. The default sampling rate with which librosa reads the file is 22050. The sample rate differs by the library you choose.

2-D Array – The first axis represents recorded samples of amplitude. And the second axis represents the number of channels. There are different types of channels – Monophonic(audio that has one channel) and stereo(audio that has two channels).

we load the data with librosa, then it normalizes the entire data and tries to give it in a single sample rate. The same we can achieve using scipy python library also. It will also give us two pieces of information – one is sample rate, and the other is data.

When you print the sample rate using scipy-it is different than librosa. Now let us visualize the wave audio data. One important thing to understand between both is- when we print the data retrieved from librosa, it can be normalized, but when we try to read an audio file using scipy, it can’t be normalized. Librosa is now getting popular for audio signal processing because of the following three reasons.

1. It tries to converge the signal into mono(one channel).
2. It can represent the audio signal between -1 to +1(in normalized form), so a regular pattern is observed.
3. It is also able to see the sample rate, and by default, it converts it to 22 kHz, while in the case of other libraries, we see it according to a different value.

**Imbalance Dataset check:**

Now we know about the audio files and how to visualize them in audio format. Moving format to data exploration we will load the CSV data file provided for each audio file and check how many records we have for each class.

The data we have is a filename and where it is present so let us explore 1st file, so it is present in fold 5 with category as a dog bark. Now use the value counts function to check records of each class.

When you see the output so data is not imbalanced, and most of the classes have an approximately equal number of records

## Data Preprocessing:

Some audios are getting recorded at a different rate-like 44KHz or 22KHz. Using librosa, it will be at 22KHz, and then, we can see the data in a normalized pattern. Now, our task is to extract some important information, and keep our data in the form of independent(Extracted features from the audio signal) and dependent features(class labels). We will use Mel Frequency Cepstral coefficients to extract independent features from audio signals.

**MFCCs –**The MFCC summarizes the frequency distribution across the window size. So, it is possible to analyze both the frequency and time characteristics of the sound. This audio representation will allow us to identify features for classification. So, it will try to convert audio into some kind of features based on time and frequency characteristics that will help us to do classification.

To demonstrate how we apply MFCC in practice, first, we will apply it on a single audio file that we are already using.

Now, we have to extract features from all the audio files and prepare the dataframe. So, we will create a function that takes the filename(file path where it is present). It loads the file using librosa, where we get 2 information. First, we’ll find MFCC for the audio data, And to find out scaled features, we’ll find the mean of the transpose of an array.

 Now, to extract all the features for each audio file, we have to use a loop over each row in the dataframe. We also use the TQDM python library to track the progress. Inside the loop, we’ll prepare a customized file path for each file and call the function to extract MFCC features and append features and corresponding labels in a newly formed dataframe.

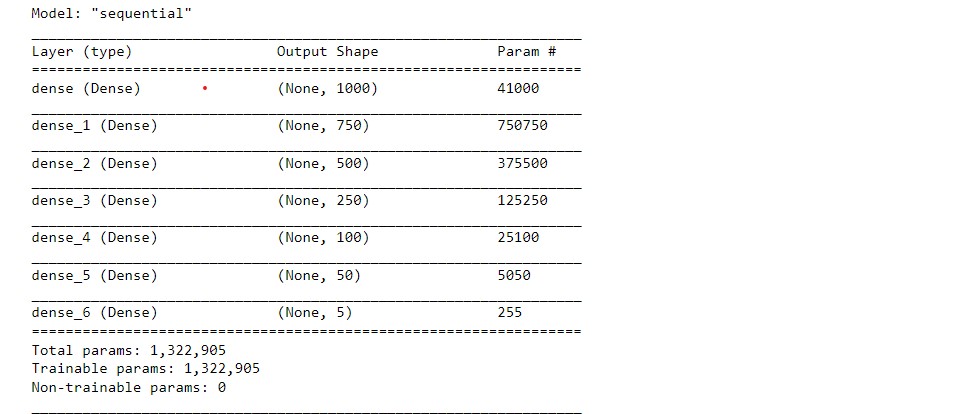
**Splitting the dataset:**

Split the dataset into train and test. 80% train data and 20% test data.

## Audio Classification Model Creation:

We have extracted features from the audio sample and splitter in the train and test set. Now we will implement an ANN model using Keras sequential API. The number of classes is 10, which is our output shape(number of classes), and we will create ANN with 3 dense layers and architecture is explained below.

1. The first layer has 100 neurons. Input shape is 40 according to the number of features with activation function as Relu, and to avoid any overfitting, we’ll use the Dropout layer at a rate of 0.5.
2. The second layer has 200 neurons with activation function as Relu and the drop out at a rate of 0.5.
3. The third layer again has 100 neurons with activation as Relu and the drop out at a rate of 0.5.



**Compile the Model**

To compile the model we need to define loss function which is categorical cross-entropy, accuracy metrics which is accuracy score, and an optimizer which is Adam.

**Train the Model**

We will train the model and save the model in HDF5 format. We will train a model for 250 epochs and batch size as 32. We’ll use callback, which is a checkpoint to know how much time it took to train over data.

**Check the Test Accuracy**

Now we will evaluate the model on test data. We got near about 97 percent accuracy on the training dataset and 100 percent on test data.

**Saving the Trained Model:**

Once you’re confident enough to take your trained and tested model into the production-ready environment, the first step is to save it into a .h5 or .pkl file using a library like pickle.

Make sure you have pickle installed in your environment.

Next, let’s import the module and dump the model into .h5 file.