Challenge

Design a machine learning model and train it with train data set of audio with and with out bird sounds, then using the model evaluate a test data set and produce a binary output based on if there is bird audio present or not.

Pre-processing of data

Pre-processing of audio files before passing them as input to machine learning models involved the following stages.

1. ***Resizing***

   We considered the audio files to be of 10 secs length, but audio files given in two data sets were of different sizes and we had to make them of same size during extraction of data from each file. So we added zeros in the end for files of length less than 10 secs and for files of lengths higher than 10 secs we truncated them in the end up to 10 secs.

   The audio files in the test data set were of sampling rate 48000 and we resampled them same as train data files with sampling rate of 44100.

2. ***Scaling***

   After we loaded the audio files and resized them to 10 secs length, we applied the rescaling of values to be in same range of (-1,1).

3. ***Mel-spectrogram Extraction***

   After loading the audio files, resizing and rescaling we have them as an one dim array of samples of the signal over time and we need to decompose the signal into its individual frequencies and frequency's amplitude. In simple words we need to convert the signal from time domain into frequency domain and this is done by extracting the mel-spectrogram of the data using the function **librosa.feature.melspectrogram**.

   Basic parameters used were suggested in Kaggle, with number of mels as 40, sampling rate as 44100, n_fft as 1024 and an overlap of 512 and maximum frequency as (sr/2). This will now give us the time frequency representation of audio data as a 2-dim array of time and frequency values of shape (40,862) where 40 is the number of mels we passed as input and 862 is calculated by the formula

   sampling rate x audio length /hop_length

   44100 X 10 / 512 = 862

4. ***Reordering data***

   To input the data to machine learning models we need them to be reordered in (time X frequency) shape which in our case is (40,862) to (862,40) and then added a pseudo column of 1s as (862,40,1) as the CNN networks needs a 3D input format.

Best performing Model Architecture

After we created the mels-sprectrogram of train and test datasets we tested the data with basic CNN models which only yielded accuracy scores around **0.59**. Many versions of standard CNNs

were tested and we found the performance of network degraded as we increased number of layers, probably due to increased difficulty to train very deep.

We then came up with the idea of creating denseNets with multiple CNNs in each block called as 'dense blocks'. So in standard CNNs if we have n layers then the output of each layer is a function of output from previous layer i,e $x_n = f(x_{n-1})$ where as in the denseNets the output of each layer is a function of outputs of all previous layers i,e $x_n = f([x_0,x_1,....x_{n-1}])$. In DenseNets the outputs of the preceding convolution layers are concatenated rather than summed to the current layer output, within each of the "dense blocks".

Our best performing denseNet architecture comprised of an initial convolution layer (Conv) and Maxpool2D layer followed by three dense blocks with 6,12,24 batchnormalization/ReLu/Conv layers and each dense block is followed by a transition block comprised of a drop out layer of 0.5, 1 set batchnormalization/ReLu/Conv layers and an AvgPool2D layer. In the dense blocks, squared 1 × 1 convolution, 3x3 convolution were used and the outputs were concatenated and it is looped for n repetitions specified for each block (6,12,24).Output from the final transition block passed to a global mean-pooling layer, which is in turn passed to dense layers of 1000 nodes and finally a dense output layer with a Softmax non-linearity activation function. The total number of layers and parameters are 89 and 5295K, respectively.

Conv2D Layer – Initial convolutional layer with input shape (862,40,1)
Maxpool2D layer
denseBlock -1 6 x [BN-ReLu-Conv, BN-ReLu-Conv, Concatenate]
transtitionBlock-1 1 x [Drop, BN-ReLu-Conv,AvgPool2D]
denseBlock -2 12 x [BN-ReLu-Conv, BN-ReLu-Conv, Concatenate]
transtitionBlock-2 1 x [Drop, BN-ReLu-Conv,AvgPool2D]
denseBlock -3 24 x [BN-ReLu-Conv, BN-ReLu-Conv, Concatenate]
transtitionBlock-2 1 x [Drop, BN-ReLu-Conv,AvgPool2D]
GlobalAvgPool2D layer
Dense layer – 1000 nodes with ReLu activation
Dense output layer – 2 nodes with softmax activation

Setup & Observations

- The whole dataset of 15690 data items were transformed as (15690,862,40,1) was divided in to train and test data sets by 80 -20 split and used as input to denseNet.
- Though we aimed to run for 100 epochs with a batch size of 32, the accuracy and val_loss saturated at around 50 epochs and hence we stopped the training at 51 epochs to avoid overfitting.
- The trained model was then used predict the whole test data set of 4512 data items in the format of (4512,862,40,1) and the best results score achieved was **0.69597**