AUDIO EVENT DETECTION

TABLE OF CONTENTS

INTRODUCTION

What is deep learning?

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network.

How deep learning works ?

Deep learning has evolved hand-in-hand with the digital era, which has brought about an explosion of data in all forms and from every region of the world. This data, known simply as big data, is drawn from sources like social media, internet search engines, e-commerce platforms, and online cinemas, among others. This enormous amount of data is readily accessible and can be shared through fintech applications like cloud computing. However, the data, which normally is unstructured, is so vast that it could take decades for humans to comprehend it and extract relevant information. Companies realize the incredible potential that can result from unraveling this wealth of information and are increasingly adapting to AI systems for automated support.

Deep learning in audio sounds

Transmitting sound through a machine and expecting an answer is a human depiction is considered as an highly-accurate deep learning task. Every one of us has come across smartphones with mobile assistants such as Siri, Alexa or Google Assistant. These are dominating and in a way invading human interactions.

Auditory information helps people recognize their surroundings. In an emergency situation, auditory information becomes even more important as it allows nearby people to react

effectively and quickly. Rare sound event detection(RSED) is a set of algorithms that aim to automatically detect certain emergency sounds with high accuracy.

Recent approaches use deep learning-based methods using deep neural network (DNN) , convolution neural network (Conv Net), recurrent neural network (RNN), and convolution recurrent neural network (CRNN). In this project, we apply a hybrid neural network of 1D Conv Net and RNN with long short-term memory units (LSTM). Frame-wise log-amplitude Mel-spectrogram is fed into our proposed model, and the model returns the output for every incoming sequence. It makes possible to estimate a relatively accurate onset time by maintaining small temporal resolution. This single model is applied to compute event probability for all three target events.

DATA COLLECTION AND VISUALISATION

We collected our data from FSDKaggle2018 data set.

Our objective is to predict the audio classification label for the audio files in the audio test folder and audio recorded from microphone.

File description

- train.csv - ground truth labels and other information relating to the training audio files (see Data Fields below)
- audio_train.zip - a folder containing the audio (.wav) training files
- audio_test.zip - a folder containing the audio (.wav) test files

Data fields

Each row of the `train.csv` file contains the following information:

- `fname`: the file name
- `label`: the audio classification label (ground truth)
- `manually_verified`: Boolean (1 or 0) flag to indicate whether or not that annotation has been manually verified; see below for additional background

About this dataset

Freesound Dataset Kaggle 2018 (or FSDKaggle2018 for short) is an audio dataset containing 18,873 audio files annotated with labels from Google's AudioSet Ontology

FSDKaggle2018 sounds are unequally distributed in the following 41 categories of the AudioSet Ontology

```
"Acoustic_guitar", "Applause", "Bark", "Bass_drum", "Burping_or_eructation", "Bus", "Cello", "Chime",
"Clarinet", "Computer_keyboard", "Cough", "Cowbell", "Double_bass", "Drawer_open_or_close",
"Electric_piano", "Fart", "Finger_snapping", "Fireworks", "Flute", "Glockenspiel", "Gong",
"Gunshot_or_gunfire", "Harmonica", "Hi-hat", "Keys_jangling", "Knock", "Laughter", "Meow",
"Microwave_oven", "Oboe", "Saxophone", "Scissors", "Shatter", "Snare_drum", "Squeak", "Tambourine",
"Tearing", "Telephone", "Trumpet", "Violin_or_fiddle", "Writing"
```

- The train set is meant to be for system development and includes ~9.5k samples unequally distributed among 41 categories.
- Out of the ~9.5k samples from the train set, ~3.7k have manually-verified ground truth annotations and ~5.8k have non-verified annotations.
- The test set is composed of ~1.6k samples with manually-verified annotations and with a similar category distribution than that of the train set.
- All audio samples in this dataset have a single label (i.e. are only annotated with one label).
- Unequal distribution of 9.6k training data samples into various classes post a challenge of having less data and a heavy model with large number of parameter to train. Thus we used “Data Augmentation techniques” to increase the number of data samples. Also we used “class rates” to fight inequality in the distribution of the data samples, we were provided with.

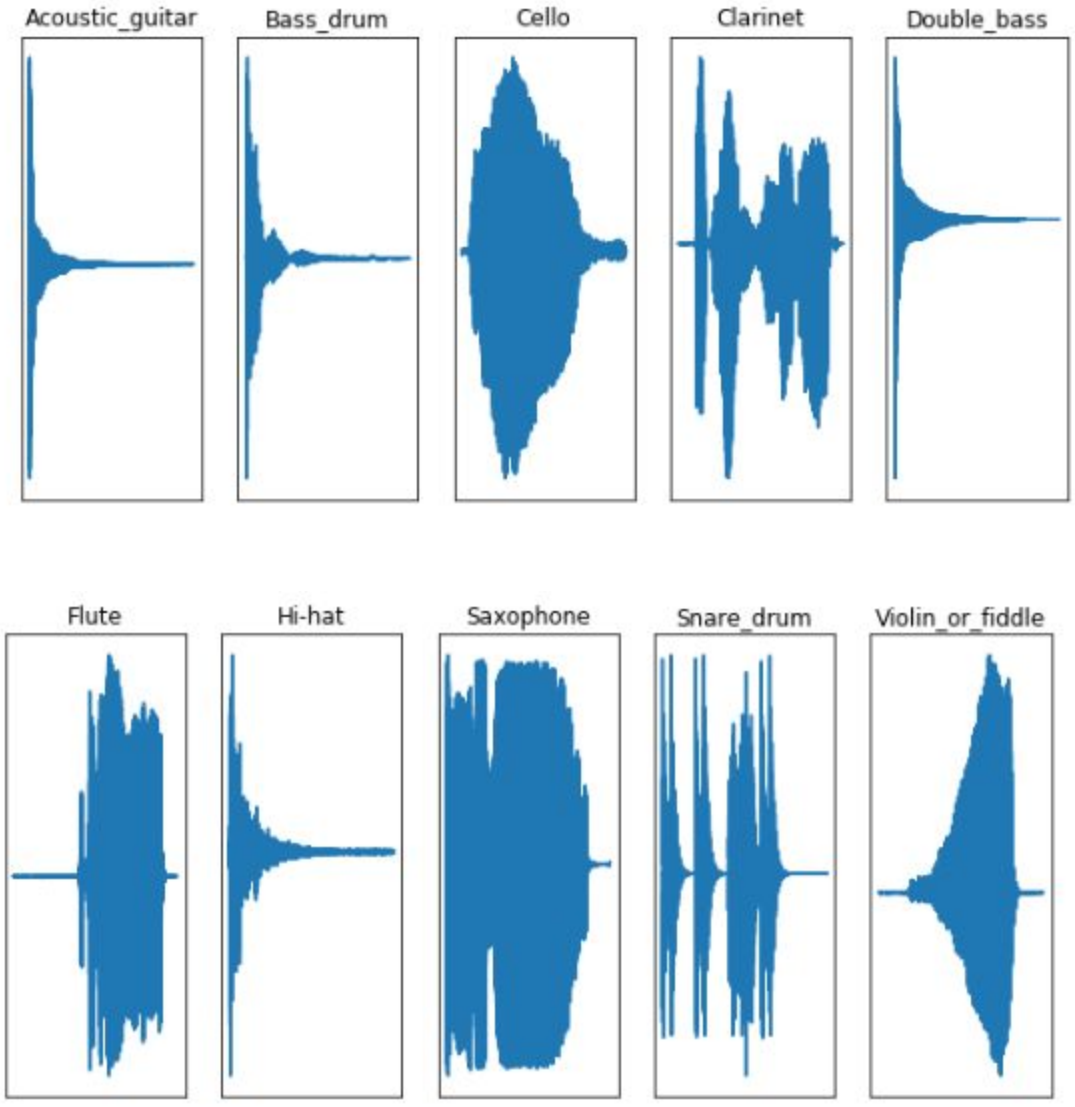
  For data augmentation we took a random index from the numpy array of signal(audio file) and took a sample of length (= rate/10) from that random index, thus taking out multiple samples from a single audio file.

  Analysing past projects on audio classification and detection we needed a ratio of 1:200 of training data samples(data samples we have) and number of samples after data augmentation(data samples we want) which meant that we should have at least 1900000 data samples to train our model to the best as we had 9.5k training samples.

  To augment such a large amount of data is very difficult with very less computational power of our laptop.Therefore,we limited our application of the detection of audio samples specific to various instruments audio clips present in FSDKaggle2018 dataset.

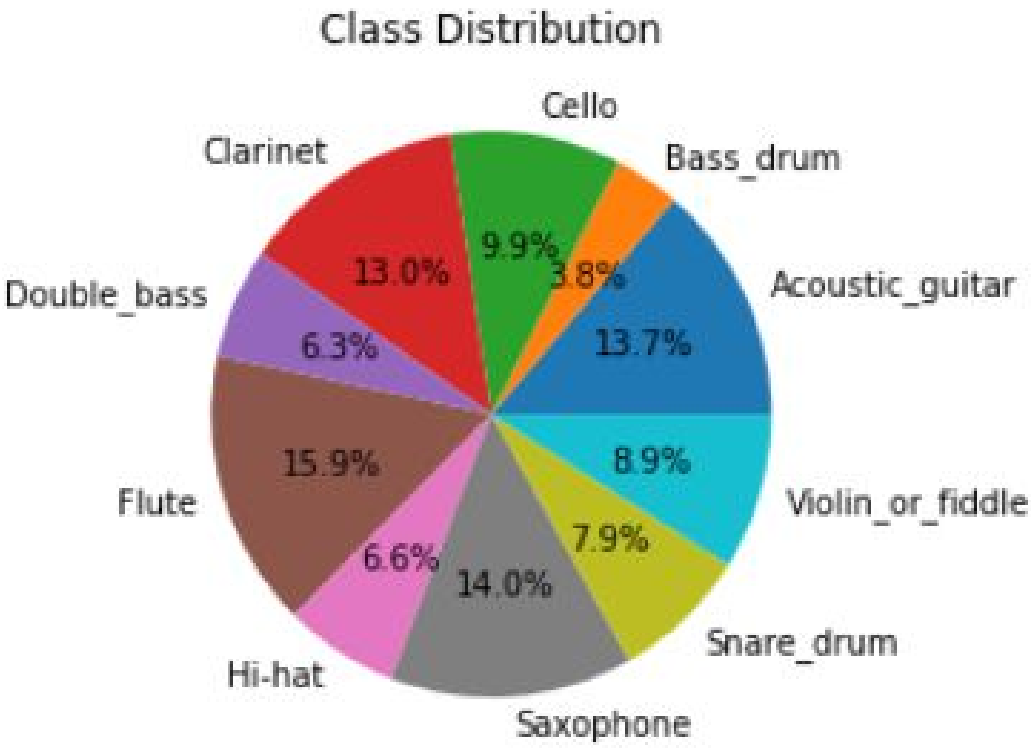
How the data looks like?

The below images represents time series of a single example from each class(A total of 10 classes)



Class distribution

The unequal distribution of training samples among 10 classes looks like below:-



The mean duration of samples in each class is depicted below:

label	
Acoustic_guitar	6.948667
Bass_drum	1.937333
Cello	5.000667
Clarinet	6.596000
Double_bass	3.206000
Flute	8.054667
Hi-hat	3.357333
Saxophone	7.124000
Snare_drum	3.987333
Violin_or_fiddle	4.530000

PREPROCESSING: FEATURE ENGINEERING

Speech processing plays an important role in any speech system whether its Automatic Speech Recognition (ASR) or speaker recognition or something else. Mel-Frequency Cepstral Coefficients (MFCCs) were very popular features for a long time; but more recently, filter banks are becoming increasingly popular. In this post, I will discuss filter banks and MFCCs and why are filter banks becoming increasingly popular. Computing filter banks and MFCCs involve somewhat the same procedure, where in both cases filter banks are computed and with a few more extra steps MFCCs can be obtained. In a nutshell, a signal goes through a pre-emphasis filter; then gets sliced into (overlapping) frames and a window function is applied to each frame; afterwards, we do a Fourier transform on each frame (or more specifically a Short-Time Fourier Transform) and calculate the power spectrum; and subsequently compute the filter banks. To obtain MFCCs, a Discrete Cosine Transform (DCT) is applied to the filter banks retaining a number of the resulting coefficients while the rest are discarded. A final step in both cases, is mean normalization.

Under Sampling

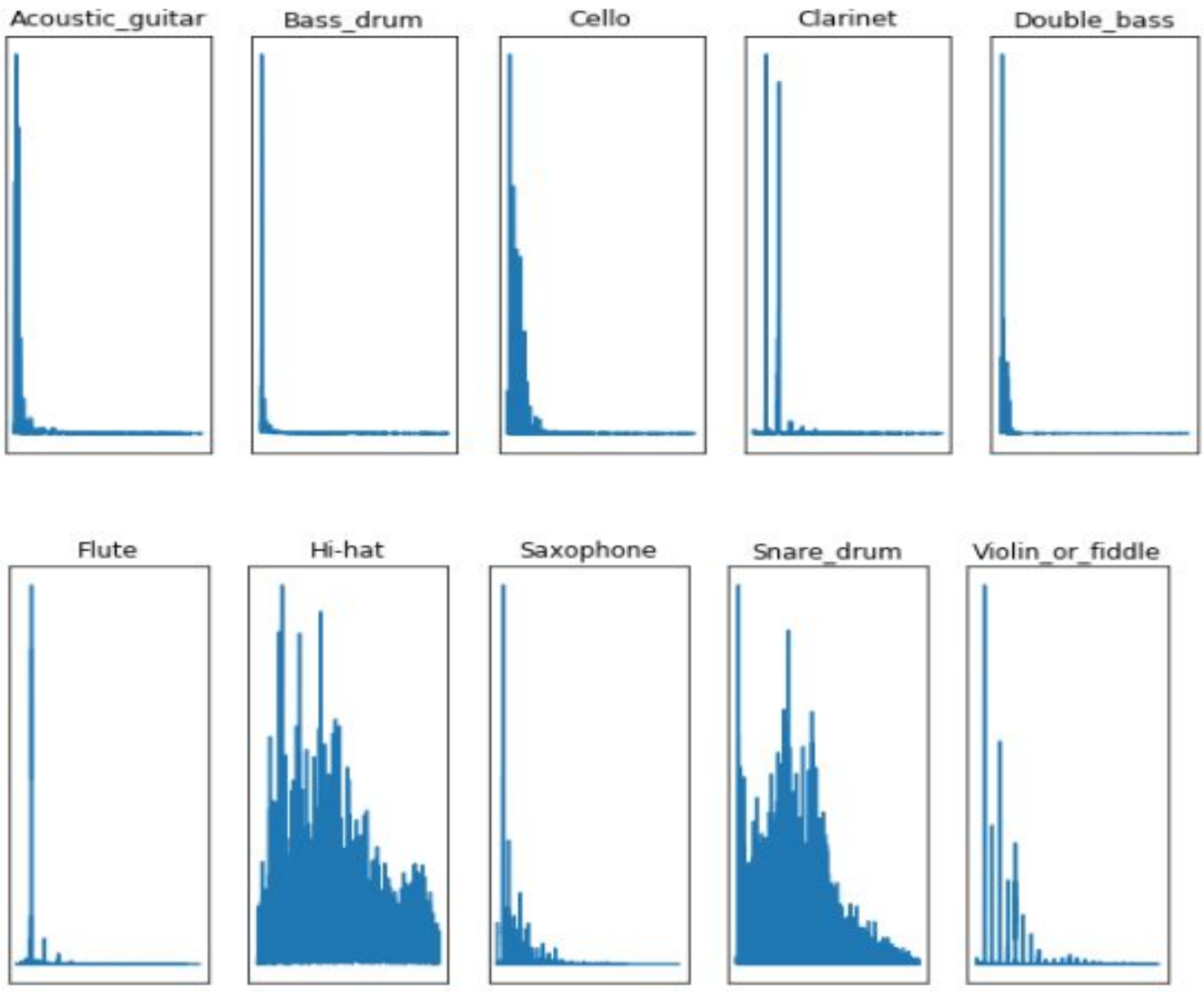
The audio signal is undersampled by decreasing its sampling rate, thus emphasising on the lower frequency components rather than the higher ones. The high frequency component does not matter much because the human ear is insensitive to the change in higher frequencies.

Also the signal is passed through a pre-emphasis filter that amplifies higher frequency components (that usually have smaller magnitude than the lower frequency components) so as to balance out frequency spectrum and may also improve SNR.

In our case , we reduced the sampling rate from the original 44.1kHz to 16kHz.

Fast Fourier Transform

A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa.

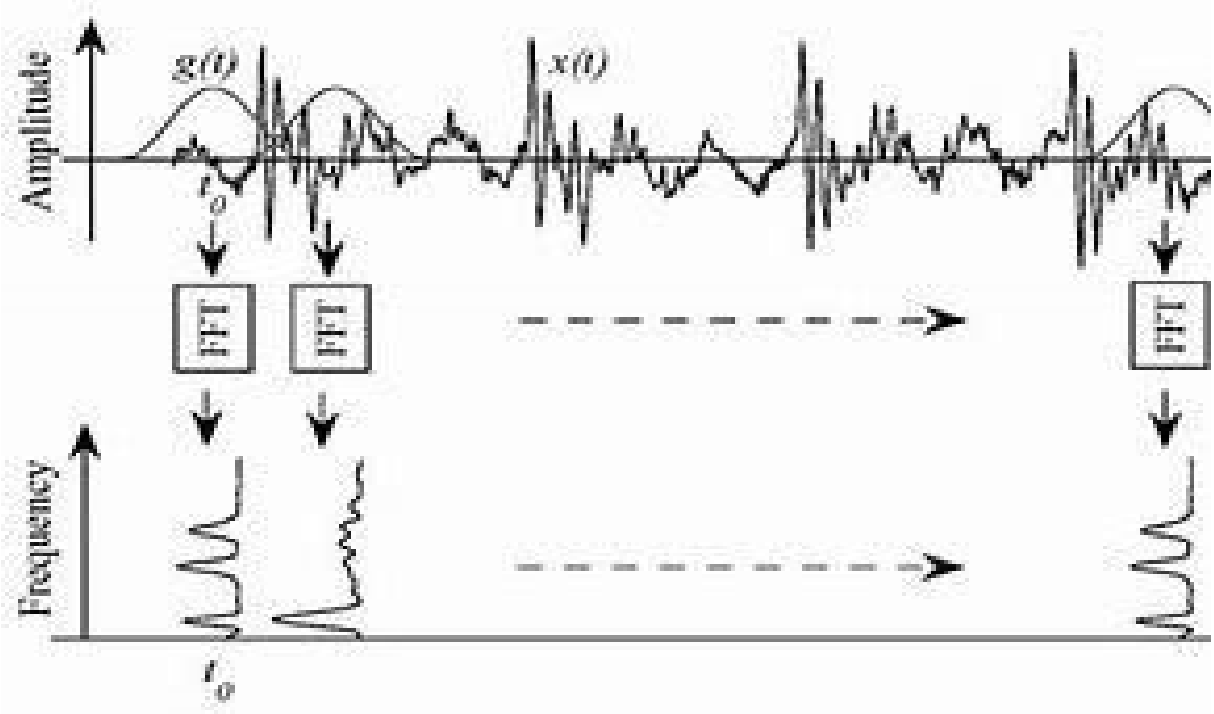


FFT encounters the problem of “spectral leakage”. Spectral leakage happens when there is a non integral frequency component in the input signal which disturbs the spectrum of the input. In other words, the number of cycles of periods in a given window length is not an integer then a spectral leakage happens. Using weighted functions called windows helps reduce spectral leakage by smoothing the sharp points at non integral frequencies.

The solution to spectral leakage and a better alternative to FFT is “Short Time Fourier Transform”.

Short Time Fourier Transform(STFT)

The Short-Time Fourier Transform (STFT), is a Fourier-related transform used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time. In practice, the procedure for computing STFTs is to divide a longer time signal into shorter segments of equal length and then compute the Fourier transform separately on each shorter segment. This reveals the Fourier spectrum on each shorter segment. One then usually plots the changing spectra as a function of time.



In our case we are slicing audio signals into overlapping frames and applying window function to each frame. Now we do fourier transform to each frame (or short time fourier transform) and evaluate its “periodogram estimate”. Converting the frequency components into Mel scale and passing it through Mel filterbanks gives us Mel filterbank coefficients. Taking logarithm of the results of each frame gives log Mel filterbank coefficients.

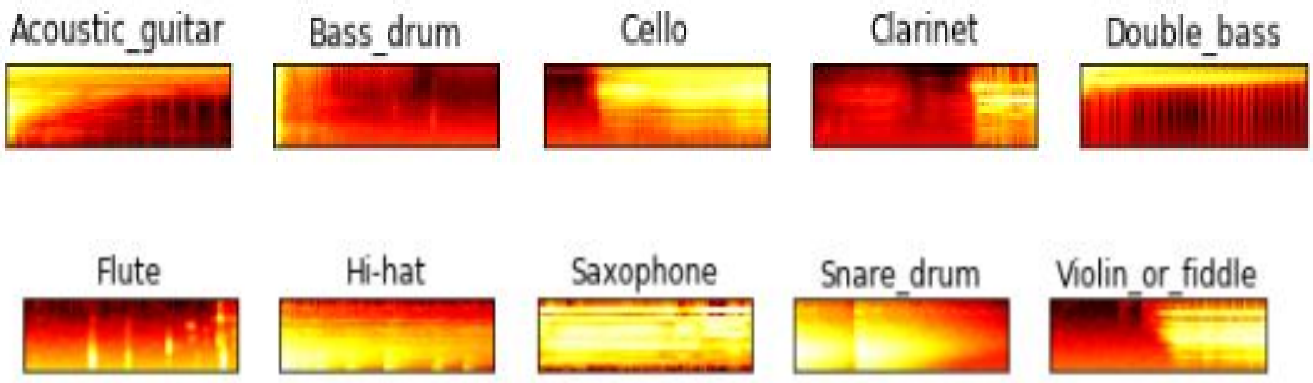
The specifications of log filterbank function includes:-

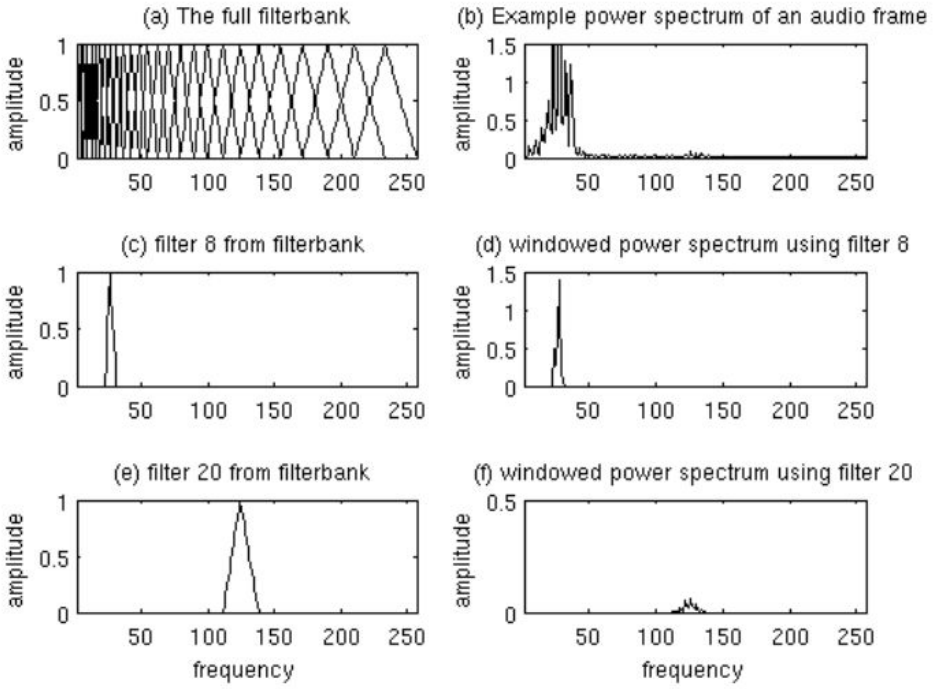
- Window length= 25ms
- Step size = 10ms
- N-FFT = 512 samples

Filter Bank Coefficients

Firstly we will convert the frequencies on the xaxis into Mel scale.

The Mel scale relates perceived frequency, or pitch, of a pure tone to its actual measured frequency. Humans are much better at discerning small changes in pitch at low frequencies than they are at high frequencies. Incorporating this scale makes our features match more closely what humans hear.





Plot of Mel Filterbank and windowed power spectrum

Applying DCT(Discrete Cosine Transformation) to the filterbank coefficients gives Mel Frequency Cepstral Coefficients(MFCCs)

MFCC(Mel Filter Cepstral Coefficient)

The first step in any automatic speech recognition system is to extract features i.e. identify the components of the audio signal that are good for identifying the linguistic content and discarding all the other stuff which carries information like background noise, emotion etc.

The main point to understand about the speech is that the sounds generated by a human are filtered by the shape of the vocal tract including tongue, teeth etc. This shape determines what sound comes out. If we can determine the shape accurately, this should give us an accurate representation of the phoneme being produced. The shape of the vocal tract manifests itself in the envelope of the short time power spectrum, and the job of MFCCs is to accurately represent this envelope.

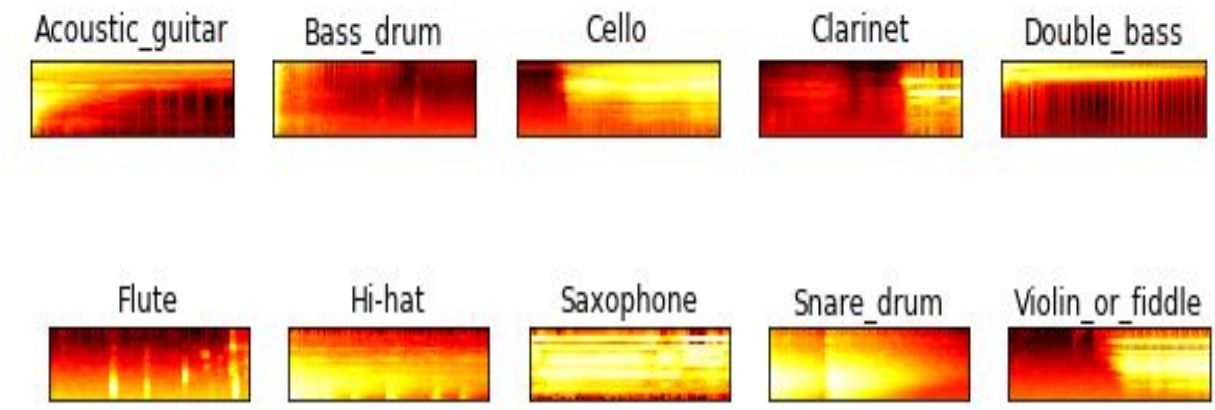
Applying DCT(Discrete Cosine Transformation) to the filterbank coefficients gives Mel Frequency Cepstral Coefficients(MFCCs).

Discrete Cosine Transform

DCTs are used to convert data into the summation of a series of cosine waves oscillating at different frequencies (more on this later). They are widely used in image and audio compression. They are very similar to Fourier Transforms, but DCT involves the use of just Cosine functions and real coefficients, whereas Fourier Transformations make use of both Sines and Cosines and require the use of complex numbers. DCTs are simpler to calculate. Both Fourier and DCT convert data from a spatial-domain into a frequency-domain and their respective inverse functions convert things back the other way.

Why are DCTs so useful? As mentioned above they are used extensively in image and audio compression. To compress analog signals, often we discard information (called lossy compression), to enable efficient compaction. We have to be careful about what information in a signal we should discard (or smooth out) when removing bits to compress a signal. DCT helps with this process. Thankfully, our eyes, ears and brain are analog devices and we are less sensitive to distortion around edges, and we are less likely to notice subtle differences fine textures.

Also, for many audio signals and graphical images the amplitudes and pixels are often similar to their near neighbors. These factors provide a solution; if we are careful at removing the higher-frequency elements of an analog signal (things that change between short ‘distances’ in the data) there is a good chance that, if we don’t take this too far, our brains might not perceive a difference.



Discrete Cosine Transformation gives us a set of coefficients or weights with increasing order of frequencies. This helps in efficient compression of signal by truncating or discarding higher frequency components of signals.

The specifications of MFCC function includes:-

- Sampling rate = 16kHz
- Window length = 25ms
- Step size = 10ms
- Number of cepstral coefficients = 13
- Number of filters = 26
- N-FFT = 512

Thus we use MFCCs as the features of input image. These features are a matrix of shape (NumOfFramesxNumOfCepstralCoefficients).

PROPOSED METHODS

Convolutional Neural Networks

- CNNs (convolutional neural networks) essentially have three parts, convolution layers, pooling layers, and fully-connected layers. It usually takes a 2D (sometimes more dimensions) matrix and outputs a result.
- Convolution starts at the top left and takes a small window with a certain width and height and performs an operation on that, the operation is usually a matrix multiplication where the matrix to multiply by is decided via gradient descent to get the best final results. It then moves according to a stride parameter and does the same. It does this all the way across the image and outputs a new image.
- Pooling is similar in the sense that it breaks the image down using small windows; however, the operation it runs on this small window is usually (average, max, or min) to combine the small window into a single pixel.
- After a set amount of convolutions and pooling, the final output is put through a fully connected layer, which is a conventional feed forward neural network to output a result.
- In our application we used a relatively small CNN model inspired by VGG-16 network with loss function as " categorical_crossentropy" and "ADAM" as an optimizer.

Case 1:Working on data with 41 classes

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 13, 9, 16)	160
conv2d_11 (Conv2D)	(None, 13, 9, 32)	4640
conv2d_12 (Conv2D)	(None, 13, 9, 64)	18496
conv2d_13 (Conv2D)	(None, 13, 9, 128)	73856
conv2d_14 (Conv2D)	(None, 13, 9, 256)	295168
max_pooling2d_2 (MaxPooling2	(None, 6, 4, 256)	0
dropout_2 (Dropout)	(None, 6, 4, 256)	0
flatten_2 (Flatten)	(None, 6144)	0
dense_4 (Dense)	(None, 6)	36870
dense_5 (Dense)	(None, 41)	287

Total params: 429,477
Trainable params: 429,477
Non-trainable params: 0

Observations

- DATA SAMPLES =14983 ,EPOCHS=10 , FEATURES MATRIX = 13X9 , ACCURACY=6.37%

```
14983/14983 [==============================] - 26s 2ms/sample - loss: 3.5863 - acc: 0.0637
```

The accuracy comes to be unexpectedly worst. This maybe due to less number of training examples and a very large number of parameters to train. Therefore,we decided to change our CNN model to have less parameters and produced more number of data samples to train.

Below is the revised CNN architecture used:

```
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 26, 19, 16)        160
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 26, 19, 16)        2320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 9, 16)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 13, 9, 32)         4640
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 13, 9, 32)         9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 4, 32)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 4, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 768)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                49216
_________________________________________________________________
dense_3 (Dense)              (None, 41)                2665
=================================================================
Total params: 68,249
Trainable params: 68,249
Non-trainable params: 0
```

- DATA SAMPLES =39670,EPOCHS=50 , FEATURES MATRIX = 26x19, ACCURACY=54.44%

  VALIDATION ACCURACY = 53.97%

```
Epoch 00050: val_acc improved from 0.53882 to 0.53973, saving model to C:\Audio-Classification-master\models\CNN
39670/39670 [==============================] - 38s 957us/sample - loss: 1.5554 - acc: 0.5444 - val_loss: 1.6445 - val_acc: 0.
5397
```

Case 2:Working on data with 10 classes

CNN Architecture used:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_28 (Conv2D)           (None, 13, 9, 16)         160
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 13, 9, 32)         4640
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 6, 4, 32)          0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 6, 4, 64)          18496
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 6, 4, 128)         73856
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 6, 4, 256)         295168
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 3, 2, 256)         0
_________________________________________________________________
dropout_6 (Dropout)          (None, 3, 2, 256)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 1536)              0
_________________________________________________________________
dense_12 (Dense)             (None, 64)                98368
_________________________________________________________________
dense_13 (Dense)             (None, 10)                650
=================================================================
Total params: 491,338
Trainable params: 491,338
Non-trainable params: 0
_________________________________________________________________
```

Observations:

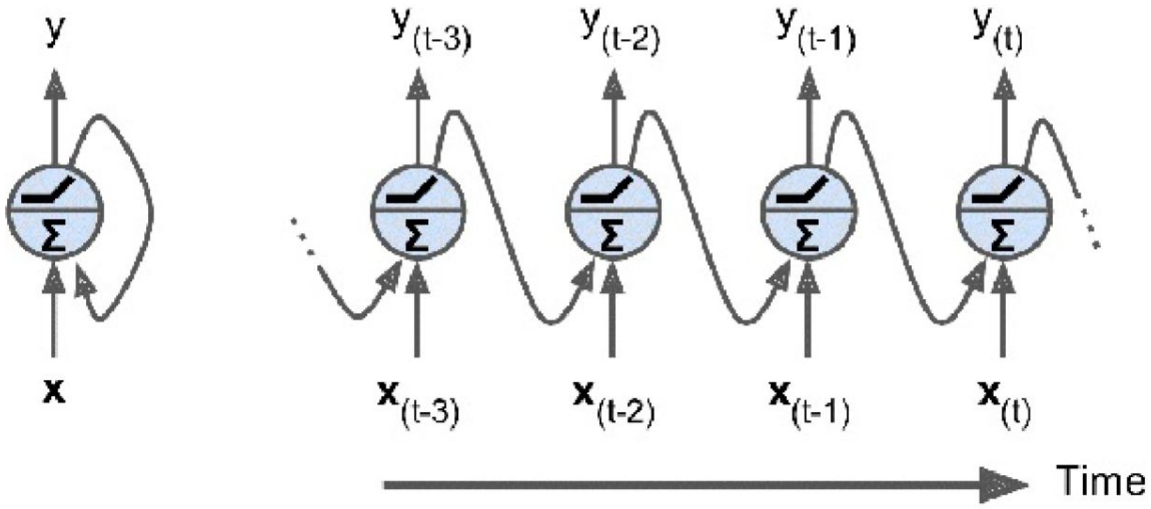
- DATA SAMPLES =57000 ,EPOCHS=10 , FEATURES MATRIX = 13X9 , ACCURACY=98.11%,

  VALIDATION ACCURACY=97.03%

```
Epoch 00010: val_acc did not improve from 0.97800
57000/57000 [==============================] - 114s 2ms/sample - loss: 0.0548 - acc: 0.9811 - val_loss: 0.1032 - val_acc: 0.9
703
```

Recurrent Neural Network(RNN)

RNNs (Recurrent Neural Networks) are a class of networks that help predicting the future.They can analyze time-series data such as stock prices and predict when to sell or buy.They can take sentences,documents or 'audio samples' as input sequences and output a sequence or a vector as per the application.It can have a sequence-to-vector network called encoder followed by a vector-to-sequences network called decoder , used for translating languages.



Case 1:Working on data with 41 classes

Below is the model we used in our application:

Layer (type)	Output Shape	Param #
lstm_10 (LSTM)	(None, 9, 128)	72704
lstm_11 (LSTM)	(None, 9, 128)	131584
time_distributed_20 (TimeDis	(None, 9, 64)	8256
time_distributed_21 (TimeDis	(None, 9, 32)	2080
time_distributed_22 (TimeDis	(None, 9, 16)	528
time_distributed_23 (TimeDis	(None, 9, 8)	136
flatten_5 (Flatten)	(None, 72)	0
dense_34 (Dense)	(None, 40)	2920
dense_35 (Dense)	(None, 41)	1681

Total params: 219,889
Trainable params: 219,889
Non-trainable params: 0

Observation

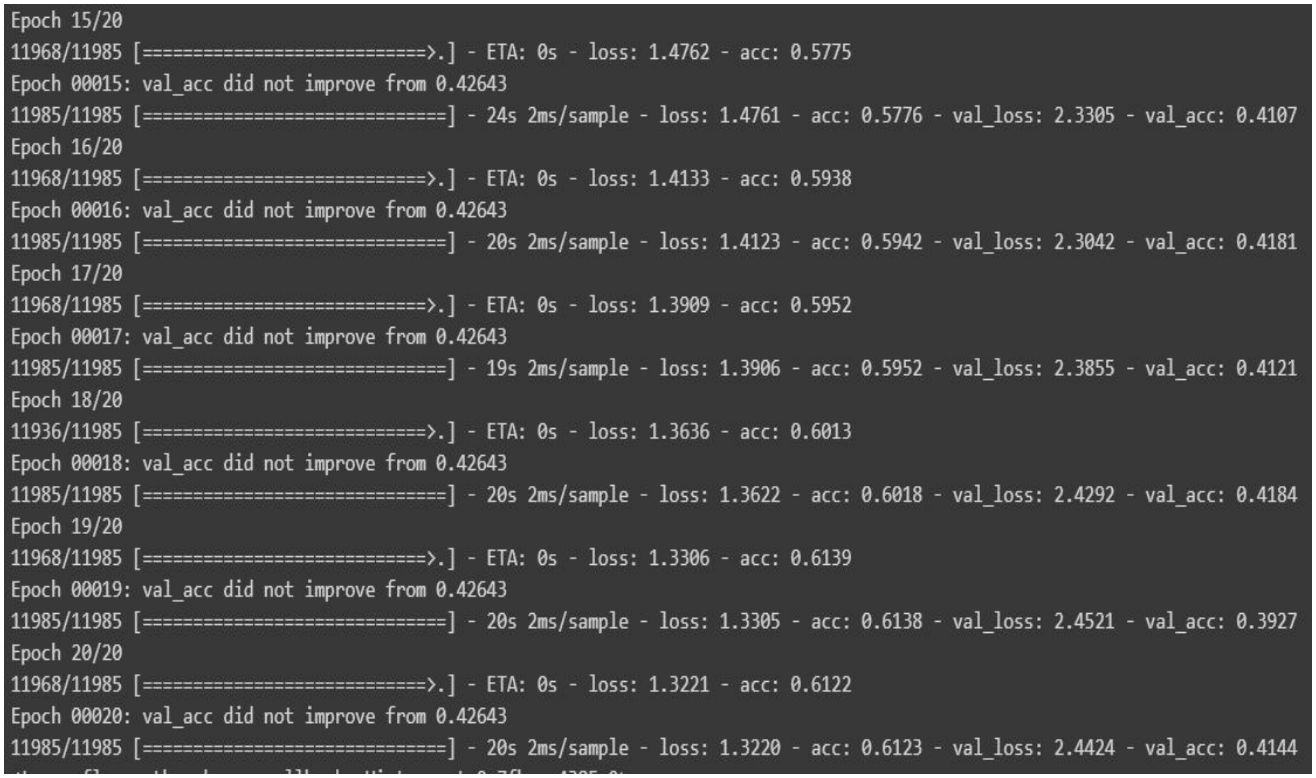
- DATA SAMPLES =13482 ,EPOCHS=10 , FEATURES MATRIX = 13X9 , ACCURACY=22.30%

```
Epoch 10/10
13440/13482 [============================>.] - ETA: 0s - loss: 2.7658 - acc: 0.2223
Epoch 00010: val_acc did not improve from 0.19693
13482/13482 [==============================] - 21s 2ms/sample - loss: 2.7643 - acc: 0.2230 - val_loss: 2.8762 - val_acc: 0.1956
```

- DATA SAMPLES = 48000 ,EPOCHS=10 , FEATURES MATRIX = 13X9 , ACCURACY=52.84%

```
Train on 7984 samples, validate on 1996 samples
Epoch 1/10
7936/7984 [============================>.] - ETA: 0s - loss: 2.1955 - acc: 0.3979
Epoch 00001: val_acc improved from -inf to 0.40030, saving model to /content/drive/My Drive/AudioEventDetection/models/RNN.model
7984/7984 [==============================] - 17s 2ms/sample - loss: 2.1941 - acc: 0.3984 - val_loss: 2.2274 - val_acc: 0.4003
Epoch 2/10
7968/7984 [============================>.] - ETA: 0s - loss: 2.0686 - acc: 0.4246
Epoch 00002: val_acc improved from 0.40030 to 0.41583, saving model to /content/drive/My Drive/AudioEventDetection/models/RNN.model
7984/7984 [==============================] - 12s 2ms/sample - loss: 2.0691 - acc: 0.4246 - val_loss: 2.1770 - val_acc: 0.4158
Epoch 3/10
7968/7984 [============================>.] - ETA: 0s - loss: 1.9948 - acc: 0.4420
Epoch 00003: val_acc improved from 0.41583 to 0.41834, saving model to /content/drive/My Drive/AudioEventDetection/models/RNN.model
7984/7984 [==============================] - 12s 2ms/sample - loss: 1.9942 - acc: 0.4421 - val_loss: 2.1462 - val_acc: 0.4183
Epoch 4/10
7968/7984 [============================>.] - ETA: 0s - loss: 1.9514 - acc: 0.4506
Epoch 00004: val_acc did not improve from 0.41834
7984/7984 [==============================] - 12s 2ms/sample - loss: 1.9515 - acc: 0.4507 - val_loss: 2.1968 - val_acc: 0.4108
Epoch 5/10
7936/7984 [============================>.] - ETA: 0s - loss: 1.8902 - acc: 0.4686
Epoch 00005: val_acc improved from 0.41834 to 0.42084, saving model to /content/drive/My Drive/AudioEventDetection/models/RNN.model
7984/7984 [==============================] - 13s 2ms/sample - loss: 1.8906 - acc: 0.4688 - val_loss: 2.1630 - val_acc: 0.4208
Epoch 6/10
7936/7984 [============================>.] - ETA: 0s - loss: 1.8703 - acc: 0.4676
Epoch 00006: val_acc did not improve from 0.42084
7984/7984 [==============================] - 12s 2ms/sample - loss: 1.8705 - acc: 0.4676 - val_loss: 2.1834 - val_acc: 0.4103
Epoch 7/10
7968/7984 [============================>.] - ETA: 0s - loss: 1.7994 - acc: 0.4887
Epoch 00007: val_acc did not improve from 0.42084
7984/7984 [==============================] - 12s 2ms/sample - loss: 1.7991 - acc: 0.4885 - val_loss: 2.1836 - val_acc: 0.4168
Epoch 8/10
7968/7984 [============================>.] - ETA: 0s - loss: 1.7568 - acc: 0.5035
Epoch 00008: val_acc did not improve from 0.42084
7984/7984 [==============================] - 12s 2ms/sample - loss: 1.7573 - acc: 0.5031 - val_loss: 2.2578 - val_acc: 0.4133
Epoch 9/10
7968/7984 [============================>.] - ETA: 0s - loss: 1.7050 - acc: 0.5104
Epoch 00009: val_acc did not improve from 0.42084
7984/7984 [==============================] - 12s 2ms/sample - loss: 1.7046 - acc: 0.5108 - val_loss: 2.2539 - val_acc: 0.3958
Epoch 10/10
7936/7984 [============================>.] - ETA: 0s - loss: 1.6620 - acc: 0.5282
Epoch 00010: val_acc did not improve from 0.42084
7984/7984 [==============================] - 12s 2ms/sample - loss: 1.6612 - acc: 0.5284 - val_loss: 2.2544 - val_acc: 0.4148
<tensorflow.python.keras.callbacks.History at 0x7fbcadaf9400>
```

- DATA SAMPLES =48000 ,EPOCHS=20 , FEATURES MATRIX = 13X9 , ACCURACY=61.23%



The model slightly overfits owing to the fact that the training accuracy is 20% more than the validation accuracy.

Case 2: Working on data with 10 classes

RNN Architecture used:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_11 (LSTM)               (None, 9, 128)            72704
_________________________________________________________________
lstm_12 (LSTM)               (None, 9, 128)            131584
_________________________________________________________________
dropout_19 (Dropout)         (None, 9, 128)            0
_________________________________________________________________
time_distributed_12 (TimeDis (None, 9, 64)             8256
_________________________________________________________________
time_distributed_13 (TimeDis (None, 9, 32)             2080
_________________________________________________________________
dropout_20 (Dropout)         (None, 9, 32)             0
_________________________________________________________________
time_distributed_14 (TimeDis (None, 9, 16)             528
_________________________________________________________________
time_distributed_15 (TimeDis (None, 9, 8)              136
_________________________________________________________________
dropout_21 (Dropout)         (None, 9, 8)              0
_________________________________________________________________
flatten_13 (Flatten)         (None, 72)                0
_________________________________________________________________
dense_39 (Dense)             (None, 40)                2920
_________________________________________________________________
dense_40 (Dense)             (None, 10)                410
=================================================================
Total params: 218,618
Trainable params: 218,618
Non-trainable params: 0
```

Observations:

- DATA SAMPLES =57000 ,EPOCHS=10 , FEATURES MATRIX = 13X9 , ACCURACY=93.69%,

  VALIDATION ACCURACY= 93.10%

```
0.2062 - acc: 0.936 - ETA: 0s - loss: 0.2061 - acc: 0.936 - ETA: 0s - loss: 0.2060 - acc: 0.936 - ETA: 0s - loss: 0.2059 - ac
c: 0.936 - ETA: 0s - loss: 0.2059 - acc: 0.936 - ETA: 0s - loss: 0.2059 - acc: 0.936 - 40s 703us/sample - loss: 0.2059 - acc:
0.9369 - val_loss: 0.2456 - val_acc: 0.9310
```

Recurrent Convolutional Neural Networks

The Convolutional Recurrent Neural Networks is the combination of two of the most prominent neural networks. The CRNN (convolutional recurrent neural network) involves CNN(convolutional neural network) followed by the RNN(Recurrent neural networks). The proposed network is similar to the CRNN but generates better or optimal results especially towards audio signal processing.

Case 1: Working on data with 41 classes

Below is the CRNN model architecture used:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_28 (Conv2D)           (None, 13, 9, 32)         320
_________________________________________________________________
batch_normalization_21 (Batc (None, 13, 9, 32)         128
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 1, 9, 32)          0
_________________________________________________________________
reshape_21 (Reshape)         (None, 9, 32)             0
_________________________________________________________________
lstm_42 (LSTM)               (None, 9, 128)            82432
_________________________________________________________________
lstm_43 (LSTM)               (None, 9, 128)            131584
_________________________________________________________________
time_distributed_33 (TimeDis (None, 9, 8)              1032
_________________________________________________________________
flatten_21 (Flatten)         (None, 72)                0
_________________________________________________________________
dense_76 (Dense)             (None, 41)                2993
=================================================================
Total params: 218,489
Trainable params: 218,425
Non-trainable params: 64
```

Observations

- DATA SAMPLES=14000 ,EPOCHS=10 , FEATURES MATRIX = 13X9 , ACCURACY=34.50% ,

  VALIDATION ACCURACY=30.42%

```
Epoch 10/10
13472/13484 [===========================>.] - ETA: 0s - loss: 2.3084 - acc: 0.3451
Epoch 00010: val_acc improved from 0.29086 to 0.30420, saving model to /content/drive/My Drive/AudioEventDetection/models/RCNN.model
13484/13484 [==============================] - 55s 4ms/sample - loss: 2.3092 - acc: 0.3450 - val_loss: 2.4994 - val_acc: 0.3042
```

- DATA SAMPLES=50000 ,EPOCHS=30 , FEATURES MATRIX = 13X9 , ACCURACY=86.80% ,

  VALIDATION ACCURACY=43.12%

```
Epoch 00025: val_acc did not improve from 0.46796
13473/13473 [==============================] - 53s 4ms/sample - loss: 0.5007 - acc: 0.8426 - val_loss: 3.4328 - val_acc: 0.4406
Epoch 26/30
13472/13473 [===========================>.] - ETA: 0s - loss: 0.4895 - acc: 0.8415
Epoch 00026: val_acc did not improve from 0.46796
13473/13473 [==============================] - 53s 4ms/sample - loss: 0.4895 - acc: 0.8415 - val_loss: 3.7884 - val_acc: 0.4119
Epoch 27/30
13472/13473 [===========================>.] - ETA: 0s - loss: 0.4701 - acc: 0.8480
Epoch 00027: val_acc did not improve from 0.46796
13473/13473 [==============================] - 53s 4ms/sample - loss: 0.4702 - acc: 0.8479 - val_loss: 3.6825 - val_acc: 0.4226
Epoch 28/30
13472/13473 [===========================>.] - ETA: 0s - loss: 0.5048 - acc: 0.8368
Epoch 00028: val_acc did not improve from 0.46796
13473/13473 [==============================] - 53s 4ms/sample - loss: 0.5049 - acc: 0.8368 - val_loss: 3.6974 - val_acc: 0.4339
Epoch 29/30
13472/13473 [===========================>.] - ETA: 0s - loss: 0.5541 - acc: 0.8288
Epoch 00029: val_acc did not improve from 0.46796
13473/13473 [==============================] - 52s 4ms/sample - loss: 0.5541 - acc: 0.8288 - val_loss: 3.6561 - val_acc: 0.4326
Epoch 30/30
13472/13473 [===========================>.] - ETA: 0s - loss: 0.4106 - acc: 0.8681
Epoch 00030: val_acc did not improve from 0.46796
13473/13473 [==============================] - 53s 4ms/sample - loss: 0.4108 - acc: 0.8680 - val_loss: 3.7642 - val_acc: 0.4312
<tensorflow.python.keras.callbacks.History at 0x7f022c68bf28>
```

Here we encounter the problem of overfitting where the bias is very less and variance is very large. In other words our model did well on training data set but bad on validation data set. To prevent this we have used L1 regularisation technique.

Also we can use dropout technique where we reduce som percentage of weights to zero.

$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n}|\theta_i|$$

- DATA SAMPLES=100000 ,EPOCHS=10 , FEATURES MATRIX = 26x19 , ACCURACY=57.53% , VALIDATION ACCURACY=52.82%

```
3ms/sample - loss: 1.4954 - acc: 0.5753 - val_loss: 1.7397 - val_acc: 0.5282
```

- DATA SAMPLES=113840 ,EPOCHS=30 , FEATURES MATRIX = 26x19 , ACCURACY=52.58% , VALIDATION ACCURACY=55.56%

```
Epoch 00030: val_acc did not improve from 0.56008
113840/113840 [==============================] - 125s 1ms/sample - loss: 1.6651 - acc: 0.5258 - val_loss: 1.5856 - val_acc:
0.5556
```

- DATA SAMPLES=180000 ,EPOCHS=50 , FEATURES MATRIX = 13X9 , ACCURACY=60.26% , VALIDATION ACCURACY=59.86%

```
[1.3835659224833772, 0.60266954]
```

- DATA SAMPLES=279633 ,EPOCHS=50 , FEATURES MATRIX = 13X9 , ACCURACY=63.93% , VALIDATION ACCURACY=61.42%

```
[1.2586375533342289, 0.6393738]
```

Case 2: Working on data with 10 classes

Below is the revised RCNN model

Layer (type)	Output Shape	Param #
conv2d_41 (Conv2D)	(None, 13, 9, 16)	160
conv2d_42 (Conv2D)	(None, 13, 9, 32)	4640
conv2d_43 (Conv2D)	(None, 13, 9, 64)	18496
batch_normalization_5 (Batch	(None, 13, 9, 64)	256
max_pooling2d_19 (MaxPooling	(None, 1, 9, 64)	0
dropout_17 (Dropout)	(None, 1, 9, 64)	0
reshape_5 (Reshape)	(None, 9, 64)	0
lstm_10 (LSTM)	(None, 9, 128)	98816
time_distributed_10 (TimeDis	(None, 9, 16)	2064
time_distributed_11 (TimeDis	(None, 9, 8)	136
dropout_18 (Dropout)	(None, 9, 8)	0
flatten_12 (Flatten)	(None, 72)	0
dense_34 (Dense)	(None, 10)	730

Total params: 125,298
Trainable params: 125,170
Non-trainable params: 128

Observations

- DATA SAMPLES=57000 ,EPOCHS=10 , FEATURES MATRIX = 13X9 , ACCURACY=97.63% ,

  VALIDATION ACCURACY=96.20%

```
57000/57000 [==============================] - ETA: 1s - loss: 0.0673 - acc: 0.976 - ETA: 1s - loss: 0.0673 - acc: 0.976 - ET
A: 1s - loss: 0.0672 - acc: 0.976 - ETA: 1s - loss: 0.0672 - acc: 0.976 - ETA: 1s - loss: 0.0672 - acc: 0.976 - ETA: 1s - los
s: 0.0673 - acc: 0.976 - ETA: 1s - loss: 0.0673 - acc: 0.976 - ETA: 1s - loss: 0.0672 - acc: 0.976 - ETA: 1s - loss: 0.0672 -
acc: 0.976 - ETA: 1s - loss: 0.0672 - acc: 0.976 - ETA: 1s - loss: 0.0672 - acc: 0.976 - ETA: 1s - loss: 0.0672 - acc: 0.976
- ETA: 1s - loss: 0.0671 - acc: 0.976 - ETA: 1s - loss: 0.0671 - acc: 0.976 - ETA: 1s - loss: 0.0671 - acc: 0.976 - ETA: 0s -
loss: 0.0671 - acc: 0.976 - ETA: 0s - loss: 0.0671 - acc: 0.976 - ETA: 0s - loss: 0.0671 - acc: 0.976 - ETA: 0s - loss: 0.067
0 - acc: 0.976 - ETA: 0s - loss: 0.0670 - acc: 0.976 - ETA: 0s - loss: 0.0670 - acc: 0.976 - ETA: 0s - loss: 0.0670 - acc: 0.
976 - ETA: 0s - loss: 0.0670 - acc: 0.976 - ETA: 0s - loss: 0.0669 - acc: 0.976 - ETA: 0s - loss: 0.0670 - acc: 0.976 - ETA:
0s - loss: 0.0671 - acc: 0.976 - ETA: 0s - loss: 0.0671 - acc: 0.976 - ETA: 0s - loss: 0.0671 - acc: 0.976 - ETA: 0s - loss:
0.0671 - acc: 0.976 - ETA: 0s - loss: 0.0671 - acc: 0.976 - ETA: 0s - loss: 0.0671 - acc: 0.976 - ETA: 0s - loss: 0.0671 - ac
c: 0.976 - 54s 953us/sample - loss: 0.0671 - acc: 0.9763 - val_loss: 0.1150 - val_acc: 0.9620
```

Results

- The testing result for the CNN model:

```
100%|██████████████████████████████████████████████████████████████████████| 300/300 [01:18<00:00,  1.80it/s]
0.9706490918844356
```

- The testing result for the RCNN output:

```
Extracting features from audio
100%|████████████████████████████████████████████████████████████| 300/300 [02:24<00:00,  2.03it/s]
0.9527933174955935
```

- The testing result for the RNN output:

  We tested this on our training set itself.

REFERENCES

- RARE SOUND EVENT DETECTION USING 1D CONVOLUTIONAL RECURRENT NEURAL NETWORKS Hyungui Lim1 , Jeongsoo Park1 ,2 , Kyogu Lee2 , Yoonchang Han1,Detection and Classification of Acoustic Scenes and Events 2017.
- CONVOLUTIONAL RECURRENT NEURAL NETWORKS FOR RARE SOUND EVENT DETECTION Emre Cakır and Tuomas Virtanen,Detection and Classification of Acoustic Scenes and Events 2017
- https://github.com/seth814/Audio-Classification
- Eduardo Fonseca, Manoj Plakal, Frederic Font, Daniel P. W. Ellis, Xavier Favory, Jordi Pons, Xavier Serra. General-purpose Tagging of Freesound Audio with AudioSet Labels: Task Description, Dataset, and Baseline. In Proceedings of DCASE2018 Workshop, 2018. URL: https://arxiv.org/abs/1807.09902
- CNN and Sequence models offered by Deeplearning.ai,Coursera
- Neural Networks and Deep Learning by Michael Nielsen