Application of Deep Learning in Urban Sounds Classification

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

SIMRITI KOUL

17BCE2211

Under the guidance of

Prof. G. Siva Shanmugam

SCOPE

VIT, Vellore.

June, 2021

DECLARATION

I hereby declare that the thesis entitled "Application of Deep Learning in Urban Sounds Classification" submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Prof. G. Siva Shanmugam.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 9th June 2021

Simriti Koul

Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled "Application of Deep Learning in Urban Sounds Classification" submitted by **Simriti Koul & 17BCE2211**, **SCOPE**, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out by her under my supervision during the period, 20.12.2020 to 20.05.2021, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 9th June 2021

Signature of the Guide

Internal Examiner **External Examiner**

Dr. VAIRAMUTHU S

ACKNOWLEDGEMENTS

This project and the research behind it would not have been possible without the exceptional support of my guide, Prof. G. Siva Shanmugam. His enthusiasm, knowledge and exacting attention to detail have been an inspiration and kept my work on track from the abstract to the final draft of this project and paper. The generosity and expertise of my guide have improved this study in innumerable ways and saved me from many errors; those that inevitably remain are entirely my own responsibility.

It can be inferred from this project that deep learning is a valuable asset in the research branch of sound classification due to the better comprehension of human mind brought about by this technological advancement. Valuable bits of knowledge regarding sound classification were feasible with Deep Learning techniques which included Multi-Layer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), understanding of normal sounds and categorizing it into classes. It is besides, assessed so that deep learning agents could improve the classification capacity of this program. In any case, there are sure challenges that may lead to constraining extent of sound classification regarding which upgrades may be available by more modification of CNNs. These hypotheses, explicitly, empower complete comprehension of sound classification other than considering challenging issues.

Studying the research papers in the domain of deep learning has proved extremely costly and I am most thankful for the guidance and support of my guide who helped this project grow. Finally, it is with true pleasure that I acknowledge the contributions of my guide who has provided responses filled with a combination of compassion and criticism that may only lead to the completion of my project.

Place : Vellore

Date : 9th June 2021

SIMRITI KOUL

Student Name

Executive Summary

Sounds are all around us. Whether directly or indirectly, we are always in contact with audio data. Sounds outline the context of our daily activities, ranging from the conversations we have when interacting with people, the music we listen to, and all the other environmental sounds that we hear on a daily basis such as a car driving past, the patter of rain, or any other kind of background noise. The human brain is continuously processing and understanding this audio data, either consciously or subconsciously, giving us information about the environment around us.

When we input sound in this program, it will automatically classify it into specific categories using Deep Learning techniques: Multi-Layer Perceptron (MLP) and modified Convolutional Neural Networks (CNN). Multi-layer perceptron’s (MLP) are classed as a type of Deep Neural Network as they are composed of more than one layer of perceptrons and use non-linear activation which distinguish them from linear perceptrons. Their architecture consists of an input layer, an output layer and in-between the two layers there is an arbitrary number of hidden layers. Whereas, Convolutional Neural Networks (CNNs) are build upon the architecture of MLPs but with a number of important changes. Firstly, the layers are organized into three dimensions, width, height and depth. Secondly, the nodes in one layer do not necessarily connect to all nodes in the subsequent layer, but often just a sub region of it.

This permits CNN to perform two critical stages. The primary being the highlight extraction stage. Here a channel window slides over the input and extricates the entirety of the convolution at each area which is at that point put away within the highlight outline. A pooling handle is regularly included between CNN layers where regularly the max esteem in each window is taken which diminishes the feature map size but retains the critical information. This can be imperative because it decreases the dimensionality of the network meaning it diminishes both the preparing time and probability of overfitting. At that point in conclusion we have the classification stage. This can be where the 3D information inside the arrangement is straightened into a 1D vector to be yield.

For the reasons examined, both MLPs and CNN’s regularly make great classifiers, where CNN’s in specific perform exceptionally well with picture classification errands due to their include extraction and classification parts. I accept that this will be exceptionally viable at finding designs inside the MFCC’s much like they are compelling at finding designs inside images.

TABLE OF CONTENTS

List of Figures

List of Tables

List of Abbreviations

MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Network
ID	Identification
MFCC	Mel-Frequency Cepstral Coefficients
D (eg.: 1D, 2D, 3D)	Dimension
SVM	Support Vector Machine
rbf	Radial Basis Function
DSP	Digital Signal Processing

Symbols and Notations

No symbols or notations used.

1. INTRODUCTION

1.1. THEORETICAL BACKGROUND

Sounds encompass us. Whether specifically or by implication, we are continuously in contact with sound information. The sounds depict the setting of our day-to-day operations, based on the discussions we have as we connected with individuals, the music we tune in to, and all the different natural sounds that we listen to on the day by day premises such as children playing, road music, motor penetrating and other sound information, either deliberately or subliminally, giving us data approximately the environment around us.

When we input sound in this program, it'll naturally classify it into particular categories utilizing Deep Learning strategies: Multi-Level Perceptron (MLP) and altered Convolutional Neural Systems (CNN). Multi-layer perceptrons (MLP) are classed as a sort of Deep Neural Networks as they are composed of more than one layer of perceptrons and utilize non-linear enactment which distinguishes them from straight perceptrons. Their engineering comprises of an input layer, a yield layer that eventually makes a forecast.

1.2. MOTIVATION

My personal motivation for working on sound classification is my interest and knowledge in DSP and Audio processing. Having worked on a number of similar projects in this field, I am keen to apply my machine learning and deep learning knowledge to this domain. I have also had knowledge of the following domains due to my peers’ excessive keenness in the field and willing to help. I have been motivated by my guide in order to complete this project with proper knowledge and understanding of the entire procedure of the program.

1.3. AIM OF THE PROPOSED WORK

The goal of this capstone project, is to apply Deep Learning techniques to the classification of environmental sounds, specifically focusing on the identification of particular urban sounds.

There is a plethora of real world applications for this research, such as:

- Content-based multimedia indexing and retrieval
- Assisting deaf individuals in their daily activities
- Smart home use cases such as 360-degree safety and security capabilities
- Automotive where recognizing sounds both inside and outside of the car can improve safety
- Industrial uses such as predictive maintenance

1.4. OBJECTIVE(S) OF THE PROPOSED WORK

The objective of this project will be to use Deep Learning techniques to classify urban sounds. For this project, we will use data set called Urbansound8K. It contains 8732 sound excerpts of urban sounds from 10 classes. Due to this project, an automatic classification of urban sounds can be made just by testing the audio clip via this program. Researchers don’t have to guess on what kind of sound they are hearing. They will get a specific result which will include its frequency-time graph and spectogram.

To achieve this, we plan on using different neural network architectures such as Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs).

2. LITERATURE SURVEY

2.1. SURVEY OF EXISTING MODELS

Title of paper referred	Authors	Year	Information Received
1) A Data set and Taxonomy for Urban Sound Research	Justin Salamon, Christopher Jacoby, Juan Pablo Bello	2014	In this paper, the authors have identified two main barriers to research in this area – the lack of a common taxonomy and the scarceness of large, real-world, annotated data. To address the first issue, the Urban Sound Taxonomy is presented, based on previous soundscape research with a focus on sound classes from real noise-complaint data. To address the second issue Urban Sound is presented, a data set containing 27 hours of audio with 18.5 hours of manually labelled sound occurrences. Through a series of classification experiments, the authors have identified avenues for future research: sensitivity to temporal scale in the analysis, confusion due to timbre similarity (especially for noise-like continuous sounds), and sensitivity to background interference.
2) Context-dependent sound event detection	Toni Heittola, Annamaria Mesaros, Antti Eronen and Tuomas Virtanen	2013	The context-dependent event priors are used to model event probabilities within the context. For example, the detection accuracy in the block-metrics is almost doubled compared to the baseline system. Furthermore, the proposed approach for detecting overlapping sound events increases the responsiveness of the sound event detection by providing better detection accuracy on the shorter 1-s blocks. Further, the event priors for the overlapping sound events are difficult to model because of high number of possible combinations and transitions between them. Latent semantic analysis has emerged as an interesting solution to learn associations between overlapping events, but the area requires more studying to apply it efficiently to the overlapping event detection.
3) A Flexible Framework for Key	Rui Cai, Lie Lu, Alan Hanjalic, Hong-Jiang Zhang,	2006	In this paper, a flexible framework has been proposed for key audio effect detection in a continuous stream and for semantic inference of related auditory context. In

Audio Effects Detection and Auditory Context Inference	and Lian-Hong Cai		this framework, two new spectral features have been introduced to improve the representation of key effects, and multi-background models are used to achieve more comprehensive characterization of the environment background. All the models are connected by the Grammar Network which represents the transition probabilities among various sounds. With this framework, an optimal key effect sequence is obtained directly from the continuous audio stream without the need of sliding window-based pre-segmentation. Based on the obtained key effect sequence, a Bayesian network-based inference has been proposed to combine the advantages of both prior knowledge and statistical learning in mapping key effects to the high-level semantics of the auditory context. Experimental evaluations have shown that the framework can achieve very satisfying results, both on key audio effect detection and semantic inference of the auditory context.
4) Unsupervised Feature Learning for Urban Sound Classification	Justin Salamon and Juan Pablo Bello	2015	In this paper the authors studied the application of unsupervised feature learning to urban sound classification. They showed that classification accuracy can be significantly improved by feature learning if they take into consideration the specificities of this domain, primarily the importance of capturing the temporal dynamics of urban sound sources.
5) Spectral vs Spectro-temporal Features for Acoustic Event Detection	Courtenay V. Cotton and Daniel P. W. Ellis	2011	Although the system is not competitive with a conventional short-frame-based system in clean conditions, it proves useful when the test data is even slightly more noisy than the training data. Features based on NMF basis activations seem to be fairly robust under moderate noise conditions (i.e. both systems using NMF features do not degrade much between 30 and 20 dB SNR). The MFCC-based system, on the other hand, performs much more poorly under moderate noise conditions. Presumably this is because the MFCC

			features are being corrupted by background noise while the NMF-based system is allowing prominent events to be represented by the same bases as they would have in the clean test data. This would therefore yield feature descriptions that are theoretically more constant as the background noise increases.
6) Classifying Soundtracks with Audio Texture Features	Daniel P. W. Ellis, Xiaohong Zeng, Josh H. McDermott	2011	In this paper, the authors have shown that the perceptually important statistics in sound textures are a useful basis for general-purpose soundtrack classification. They can be used to recognize foreground sound categories like speech, music, and clapping, as well as more loosely-defined contexts such as outdoor-rural, and indoor-noisy. Classifiers based on texture statistics are able to achieve accuracies very similar to those based on conventional MFCC features, and the two approaches can be easily and profitably combined.

2.2 Summary/Gaps Identified in Survey

There are many gaps identified in the survey like the lack of a common taxonomy and the scarceness of large, real-world, annotated data. Further, the event priors for the overlapping sound events are difficult to model because of high number of possible combinations and transitions between them. Although latent semantic analysis has emerged as an interesting solution to learn associations between overlapping events, but the area requires more studying to apply it efficiently to the overlapping event detection. There is also a research gap concerning the performance and usefulness of deep neural networks, designed for normal object recognition, when it comes to classify Spectrograms and similar sound related images.

3. OVERVIEW OF THE PROPOSED SYSTEM

3.1 INTRODUCTION AND RELATED CONCEPTS

The proposed solution to this problem is to apply Deep Learning techniques that have proved to be highly successful in the field of image classification.

First we will extract Mel-Frequency Cepstral Coefficients (MFCC) from the audio samples on a per-frame basis with a window size of a few milliseconds. 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. These audio representations will allow us to identify features for classification.

The next step will be to train a Deep Neural Network with these data sets and make predictions. We will begin by using a simple neural network architecture, such as Multi-Layer Perceptron before experimenting with more complex architectures such as Convolutional Neural Networks.

Multi-layer perceptrons (MLP) are classed as a type of Deep Neural Network as they are composed of more than one layer of perceptrons and use non-linear activation which distinguish them from linear perceptrons. Their architecture consists of an input layer, an output layer that ultimately make a prediction about the input, and in-between the two layers there is an arbitrary number of hidden layers.

These hidden layers have no direct connection with the outside world and perform the model computations. The network is fed a labeled data set (this being a form of supervised learning) of input-output pairs and is then trained to learn a correlation between those inputs and outputs. The training process involves adjusting the weights and biases within the perceptrons in the hidden layers in order to minimize the error.

The algorithm for training an MLP is known as Backpropagation. Starting with all weights in the network being randomly assigned, the inputs do a forward pass through the network and the decision of the output layer is measured against the ground truth of the labels you want to predict. Then the weights and biases are backpropagated back though the network where an optimization method, typically Stochastic Gradient descent is used to adjust the weights so they will move one step closer to the error minimum on the next pass. The training phase will keep on performing this cycle on the network until it the error can go no lower which is known as convergence.

Convolutional Neural Networks (CNNs) build upon the architecture of MLPs but with a number of important changes. Firstly, the layers are organized into three dimensions, width, height and depth. Secondly, the nodes in one layer do not necessarily connect to all nodes in the subsequent layer, but often just a sub region of it.

This allows the CNN to perform two important stages. The first being the feature extraction phase. Here a filter window slides over the input and extracts a sum of the convolution at each location which is then stored in the feature map. A pooling process is often included between CNN layers where typically the max value in each window is taken which decreases the feature map size but retains the significant data. This is important as it reduces the dimensionality of the network meaning it reduces both the training time and likelihood of over-fitting. Then lastly we have the classification phase. This is where the 3D data within the network is flattened into a 1D vector to be output.

For the reasons discussed, both MLP and CNN typically make good classifiers, where CNNs in particular perform very well with image classification tasks due to their feature extraction and classification parts. I believe that this will be very effective at finding patterns within the MFCC's much like they are effective at finding patterns within images.

3.2 FRAMEWORK, ARCHITECTURE OR MODULE FOR THE PROPOSED SYSTEM

Initial model architecture - MLP

We will start with constructing a Multilayer Perceptron (MLP) Neural Network using Keras and a Tensorflow backend.

Starting with a sequential model so we can build the model layer by layer.

We will begin with a simple model architecture, consisting of three layers, an input layer, a hidden layer and an output layer. All three layers will be of the dense layer type which is a standard layer type that is used in many cases for neural networks.

The first layer will receive the input shape. As each sample contains 40 MFCCs (or columns) we have a shape of (1x40) this means we will start with an input shape of 40.

The first two layers will have 256 nodes. The activation function we will be using for our first 2 layers is the ReLU, or Rectified Linear Activation. This activation function has been proven to work well in neural networks.

We will also apply a Dropout value of 50% on our first two layers. This will randomly exclude nodes from each update cycle which in turn results in a network that is capable of better generali- sation and is less likely to overfit the training data.

Our output layer will have 10 nodes (num_labels) which matches the number of possible classifi- cations. The activation is for our output layer is softmax. Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability.

Convolutional Neural Network (CNN) model architecture

We will modify our model to be a Convolutional Neural Network (CNN) again using Keras and a Tensorflow backend.

Again we will use a sequential model, starting with a simple model architecture, consisting of four Conv2D convolution layers, with our final output layer being a dense layer.

The convolution layers are designed for feature detection. It works by sliding a filter window over the input and performing a matrix multiplication and storing the result in a feature map. This operation is known as a convolution.

The filter parameter specifies the number of nodes in each layer. Each layer will increase in size from 16, 32, 64 to 128, while the kernel_size parameter specifies the size of the kernel window which in this case is 2 resulting in a 2x2 filter matrix.

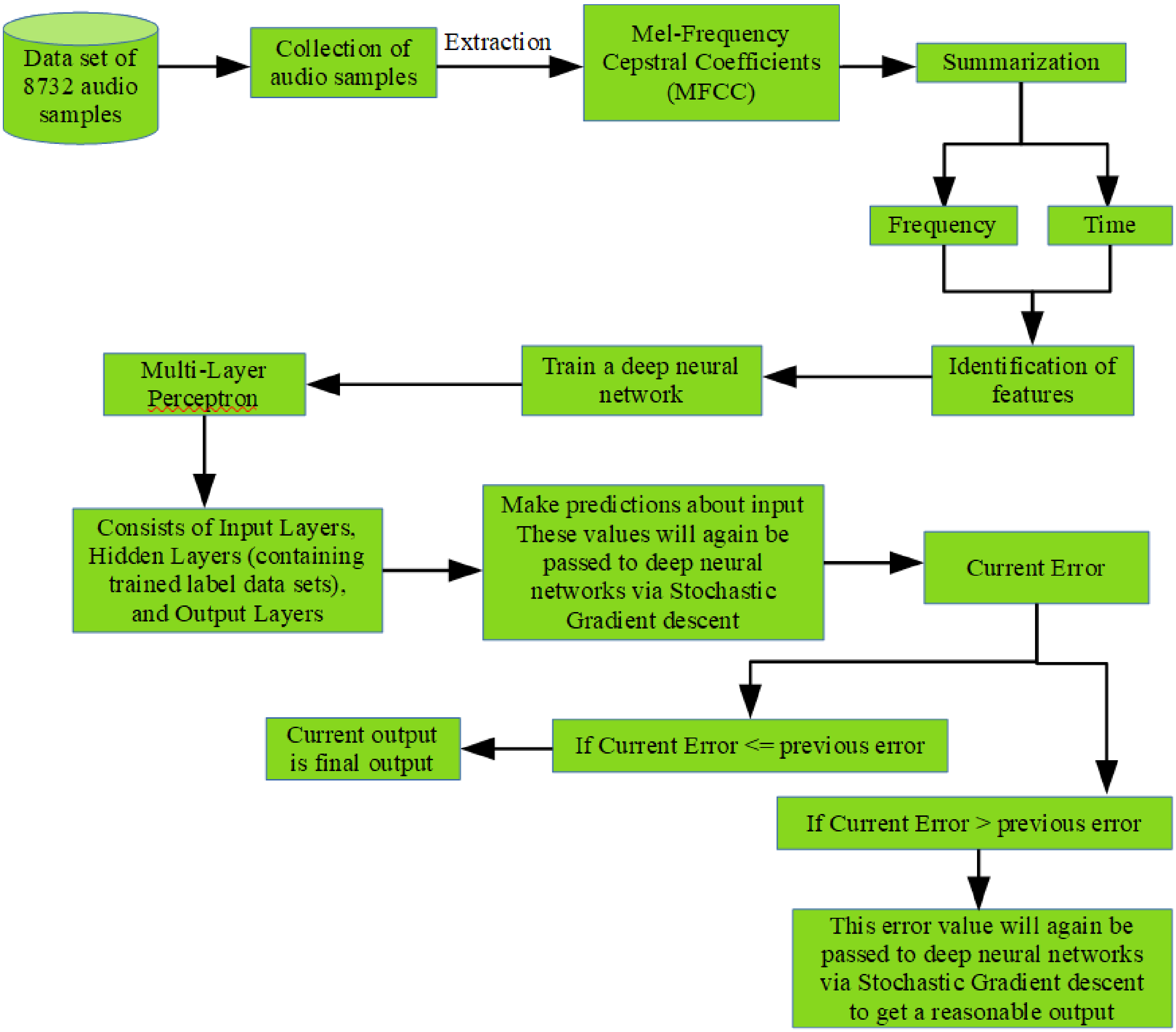
The first layer will receive the input shape of (40, 174, 1) where 40 is the number of MFCC's 174 is the number of frames taking padding into account and the 1 signifying that the audio is mono.

The activation function we will be using for our convolutional layers is ReLU which is the same as our previous model. We will use a smaller Dropout value of 20% on our convolutional layers.

Each convolutional layer has an associated pooling layer of MaxPooling2D type with the final convolutional layer having a GlobalAveragePooling2D type. The pooling layer is do reduce the dimensionality of the model (by reducing the parameters and subsquent computation require- ments) which serves to shorten the training time and reduce overfitting. The Max Pooling type takes the maximum size for each window and the Global Average Pooling type takes the average which is suitable for feeding into our dense output layer.

Our output layer will have 10 nodes (num_labels) which matches the number of possible classifi- cations. The activation is for our output layer is softmax. Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability.

3.3 PROPOSED SYSTEM MODEL (ER Diagram)



4. PROPOSED SYSTEM ANALYSIS AND DESIGN

4.1 INTRODUCTION

To analyze the proposed system, we need to take the following steps:

1. Data Preprocessing & Data Splitting

We recognized the taking after sound properties that require preprocessing to guarantee consistency over the complete dataset:

- Sound Channels
- Test rate
- Bit-depth

We will proceed to utilize Librosa which can be valuable for the pre-processing and include extraction.

A. Audio properties that require normalizing

For much of the preprocessing, we'll be able to utilize Librosa's load() function.

We will compare the yields from Librosa against the default outputs of scipy's wave file library employing a chosen record from the dataset.

B. Preprocessing stage

- **Sample rate conversion**

  By default, Librosa's load function changes over the examining rate to 22.05 kHz which we are able to utilize as our comparison level.

- **Bit Depth**

  Librosa's load function will moreover standardize the information so its values extend between -1 and 1. This evacuates the complication of the dataset having a wide extend of bit-depths.

- **Merge audio channels**

  Librosa will too change over the flag to mono, meaning the number of channels will continuously be 1.

- **Other properties**

  At this stage it is not yet clear whether other factors may also need to be taken into account, such as sample duration length and volume levels.

  We will proceed as is for the meantime and come back to address these later if it's perceived to be effecting the validity of our target metrics.

C. Extract Features

As laid out within the proposition, we are going to extricate Mel-Frequency Cepstral Coefficients (MFCC) from the sound samples.

The MFCC outlines the recurrence dispersion over the window measure, so it is conceivable to dissect both the recurrence and time characteristics of the sound. These sound representations will permit us to distinguish highlights for classification.

- **Extracting a MFCC**

  For this, we will utilize Librosa's mfcc() work which creates an MFCC from time arrangement sound information.

- **Extracting MFCC's for every file**

  We will presently extricate an MFCC for each sound record within the dataset and store it in a Panda Dataframe in conjunction with its classification name.

D. Convert the data and labels

We will utilize sklearn.preprocessing.LabelEncoder to encode the categorical content information into model-understandable numerical information.

E. Split the dataset

Here we will utilize sklearn.model_selection.train_test_split to part the dataset into preparing and testing sets. The testing set size will be 20% and we are going to set an irregular state.

System Design includes the following parts:

a. Data Exploration and Visualization

For our program, we will require a dataset and we will be referring Urbansound8K for it. The dataset contains 8732 sound excerpts (<=4s) of urban sounds from 10 classes, which are:

- Air Conditioner
- Car Horn
- Children Playing
- Dog bark
- Drilling
- Engine Idling

- Gun Shot
- Jackhammer
- Siren
- Street Music

The accompanying metadata contains a unique ID for each sound excerpt along with it's given class name. A sample of this dataset is included with the accompanying git repo and the full dataset can be downloaded from Urbansounds8K official website.

b. Algorithms and Techniques

The proposed solution to this problem is to apply Deep Learning techniques that have proved to be highly successful in the field of image classification.

First we will extract Mel-Frequency Cepstral Coefficients (MFCC) from the audio samples on a per-frame basis with a window size of a few milliseconds. The MFCC summarizes the frequency distribution across the window size, so it is possible to analyse both the frequency and time characteristics of the sound. These audio representations will allow us to identify features for classification.

The next step will be to train a Deep Neural Network with these data sets and make predictions. We will begin by using a simple neural network architecture, such as Multi-Layer Perceptron before experimenting with more complex architectures such as Convolutional Neural Networks.

Multi-layer perceptron's (MLP) are classed as a type of Deep Neural Network as they are composed of more than one layer of perceptrons and use non-linear activation which distinguish them from linear perceptrons. Their architecture consists of an input layer, an output layer that ultimately make a prediction about the input, and in-between the two layers there is an arbitrary number of hidden layers. These hidden layers have no direct connection with the outside world and perform the model computations. The network is fed a labelled dataset (this being a form of supervised learning) of input-output pairs and is then trained to learn a correlation between those inputs and outputs. The training process involves adjusting the weights and biases within the perceptrons in the hidden layers in order to minimize the error.

The algorithm for training an MLP is known as Backpropagation. Starting with all weights in the network being randomly assigned, the inputs do a forward pass through the network and the decision of the output layer is measured against the ground truth of the labels you want to predict. Then the weights and biases are backpropagated back though the network where an optimisation method, typically Stochastic Gradient descent is used to adjust the weights so they will move one step closer to the error minimum on the next pass. The training phase will keep on performing this cycle on the network until it the error can go no lower which is known as convergence.

Convolutional Neural Networks (CNNs) build upon the architecture of MLPs but with a number of important changes. Firstly, the layers are organised into three dimensions, width, height and depth. Secondly, the nodes in one layer do not necessarily connect to all nodes in the subsequent layer, but often just a sub region of it. This allows the CNN to perform two important stages. The first being the feature extraction phase. Here a filter window slides over the input and extracts a sum of the convolution at each location which is then stored in the feature map. A pooling process is often included between CNN layers where typically the max value in each window is taken which decreases the feature map size but retains the significant data. This is important as it reduces the dimensionality of the network meaning it reduces both the training time and likelihood of overfitting. Then lastly we have the classification phase. This is where the 3D data within the network is flattened into a 1D vector to be output.

For the reasons discussed, both MLPs and CNN’s typically make good classifiers, where CNN’s in particular perform very well with image classification tasks due to their feature extraction and classification parts. I believe that this will be very effective at finding patterns within the MFCC’s much like they are effective at finding patterns within images.

c. Benchmark Models

For the benchmark model, we are going to utilize the calculations sketched out within the paper "A Dataset and Scientific classification for Urban Sound Research" (Salamon, 2014). The paper depicts five distinctive calculations with the taking after exactnesses for a sound cut most extreme term of 4 seconds utilizing the same UrbanSound dataset.

Table 4.1 Classification Accuracy of Benchmark Models

Algorithm	**Classification Accuracy**
SVM_rbf	68%
RandomForest500	66%
IBk5	55%
J48	48%
ZeroR	10%

4.2 REQUIREMENT ANALYSIS

The project requires to be executed on a local machine with the availability of internet connection and data set downloaded from specified website. No other hardware requirements are needed for the execution of the project. Anaconda software was installed and Jupyter was launched on it to implement the program. Colab can also be used to run the Jupyter notebooks in less processing time. The packages used to run this program are:

1. TensorFlow
2. Keras
3. Microsoft Cognitive Toolkit
4. Librosa
5. Nolearn
6. PyLearn2
7. Numpy
8. PyTorch
9. Scikit-Learn
10. Pandas
11. Ipython

4.2.1 FUNCTIONAL REQUIREMENTS

4.2.1.1 PRODUCT PERSPECTIVE

Our program is an audio classification model built on Multi-Level Perceptron Architecture. It is coded in python. It accepts audio from user as input, which is then analyzed based on the trained data set. If there is occurance of any error, then that audio data is passed again through the model and it is trained until the error is rectified. These then trained data sets will be used to train our CNN model to classify audio.

4.2.1.2 PRODUCT FEATURES

Our model is able to distinguish between similar sounds using modified Neural Networks.

- It accepts an audio sample from the user.
- It demonstrates the confidence of audio belonging to a particular class
- It visualizes the result using frequency-time chart.
- A CNN Model of 92% accuracy on training set and 87% accuracy on testing set

4.2.1.3 USER CHARACTERISTICS

The program can be used by any user who wants to analyze and classify audio. The program is user friendly in every manner and can be used independently without any special access or permissions. The users should have basic computer skills and preferably some knowledge of python language.

4.2.1.4 ASSUMPTION & DEPENDENCIES

The program/model requires internet connection since it is hosted online but could be deployed on a local host of the user machine. In the current version of this program, it could be run using python on jupyter along with being hosted on the local host of the user machine.

4.2.1.5 DOMAIN REQUIREMENTS

The domain requirements for the application to work is the availability of a local machine inclusive of an operating system, command prompt, Python version 2.5 or later and Jupyter on Anaconda.

The model has been previously trained and conditioned to work and thus does not need to be re-trained or configured on the host machine. The administrative system requires Scikit-Learn , Keras , Anaconda and Python 3 or later versions.

4.2.1.6 USER REQUIREMENTS

The user will easily be able to differentiate or classify between sounds. The user requires a local machine on which the program can be accessed.

The end users of the application is not limited to a particular segment but can be widely used by any user who wishes to classify the audio since a lot of the tampered audio samples are nowadays used as a source of information in various fields and some of them are also used to conceal important facts.

4.2.2. NON-FUNCTIONAL REQUIREMENTS

4.2.2.1. PRODUCT REQUIREMENTS

The product needs to be executed on a system which has Anaconda installed and Jupyter Notebook launched on it. User can also use Colab to execute this program.

4.2.2.1.1. EFFICIENCY (IN TERMS OF TIME AND SPACE)

The application is highly efficient in terms of time and space. It runs with maximum duration of 4 seconds when the application loads and requires 10MB of space in the local machine when loaded.

In our initial attempt, we were able to achieve a Classification Accuracy score of:

- Training data Accuracy: 92.3%
- Testing data Accuracy: 87%

The Training and Testing accuracy scores are both high and an increase on our initial model. Training accuracy has increased by ~6% and Testing accuracy has increased by ~4%.

There is a marginal increase in the difference between the Training and Test scores (~6% com- pared to ~5% previously) though the difference remains low so the model has not suffered from overfitting.

The final model achieved a classification accuracy of 92% on the testing data which exceeded my expectations given the benchmark was 68%.

Table 4.2.2.1.1 Classification Accuracy of Final and Benchmark Models

Model	**Classification Accuracy**
CNN	92%
MLP	88%
Benchmark SVM_rbf	68%

The final solution performs well when presented with a .wav file with a duration of a few seconds and returns a reliable classification.

However, we do not know how the model would perform on Real-time audio. We do not know whether it would be able to perform the classification in a timely manner so audio frames are not skipped or the classification would be heavily affected by latency.

Also, we do not know how the classifier would perform in a real world setting. Our study makes no attempt to determine the effect of factors such as noise, echos, volume and salience level of the sample.

4.2.2.1.2. RELIABILITY

The model is reliable and performs well when presented with a .wav file with a duration of a few seconds. However, the model may not be reliable on real-time audio. It makes no attempt to determine the effect of factors such as noise, echos, volume, and salience level of the sample.

4.2.2.1.3. PORTABILITY

The model is quite portable considering a few requirements that need to be satisfied before implementing it on a separate device. The device should have the necessary platform (Jupyter) installed in it beforehand. The device must have all the libraries and packages of Python language installed and tested. The device must have microphone in-built or connected to it as, for testing, the audio will be provided using a speaker of another device. If these requirements are matched, then the model is portable and efficient to be implemented on another device.

4.2.2.1.4. USABILITY

The application can be used by any user who wishes to get an insight whether which class an audio sample lies in and wants to do an analysis on the audio file which is being used. The usability of the application is not restricted to any group but could be widely used by law officials or individuals who would want to determine the authenticity of audio.

4.2.2.2. ORGANIZATIONAL REQUIREMENTS

4.2.2.2.1. IMPLEMENTATION REQUIREMENTS

The application has an inbuilt CNN model which is trained on the Urbansound8K dataset to detect the variation between the sounds. The user machine should have Python 2.5 or later and Jupyter in Anaconda as a framework. Once deployed the host needs the internet to access the application.

4.2.2.2.2. ENGINEERING STANDARD REQUIREMENTS

The user using the application should have some knowledge of python language in order to execute the program. The program needs to be executed on a local machine with internet connection, Anaconda installed and Jupyter Notebook launched. If these requirements cannot be satisfied, then the local machine should have continuous internet connection and the program can be executed in Colab.

4.2.2.3. OPERATIONAL REQUIREMENTS

- ECONOMIC – the project does not have any economic requirements and can be executed without restrictions.
- ENVIRONMENTAL – this project is eco-friendly as it does not have any impact on the environment and takes minimum hosting space.
- SOCIAL – this project does not have any social constraints. It can be used with variety of users.
- POLITICAL – this project does not have any political contraints and can be executed without any restrictions.
- ETHICAL – the project does not have any ethical constraints. It does not hamper with any belief.
- HEALTH AND SAFETY – the project does not have any health and safety hazards and can be executed normally.
- SUSTAINABILITY – the project is sustainable and can be executed whenever required.
- LEGALITY – the project does not have any legal constraints and can be executed without any restrictions.

4.2.3. SYSTEM REQUIREMENTS

4.2.3.1. HARDWARE REQUIREMENTS

The project requires to be executed on a local machine satisfying minimum requirements like the availability of internet connection, installation of Anaconda, launching of Jupyter Notebook or executing the program on Colab. No other hardware requirements are needed for the execution of the project.

4.2.3.2. SOFTWARE REQUIREMENTS

The project requires to be executed on a local machine with the availability of internet connection and data set downloaded from specified website. No other hardware requirements are needed for the execution of the project. Anaconda software was installed and Jupyter was launched on it to implement the program. Colab can also be used to run the Jupyter notebooks in less processing time. The packages used to run this program are:

1. TensorFlow
2. Keras
3. Microsoft Cognitive Toolkit
4. Librosa
5. Nolearn
6. PyLearn2
7. Numpy
8. PyTorch
9. Scikit-Learn
10. Pandas
11. IPython

5. RESULT AND DISCUSSION

5.1. Model evaluation and validation

Amid the demonstrate advancement stage the approval information was utilized to assess the show. The ultimate demonstrate architecture and hyperparameters were chosen since they performed the leading among the attempted combinations.

As we can see from the approval work within the past area, to confirm the strength of the ultimate demonstration, a test was conducted utilizing copyright-free sounds sourced from the web. The taking after perceptions are based on the comes about of the test:

▪ The classifier performs well with modern data.

▪ Misclassification does happen but appears to be between classes that are generally comparative such as Drilling and Jackhammer.

It was previously noted in our data exploration, that it is difficult to visualize the difference between some of the classes. In particular, the following sub-groups are similar in shape:

- ▪ Repetitive sounds for air conditioner, drilling, engine idling and jackhammer.
- ▪ Sharp peaks for dog barking and gun shot.
- ▪ Similar pattern for children playing and street music.

Using a disarray network we'll look at if the ultimate demonstrate moreover battled to distinguish between these classes.

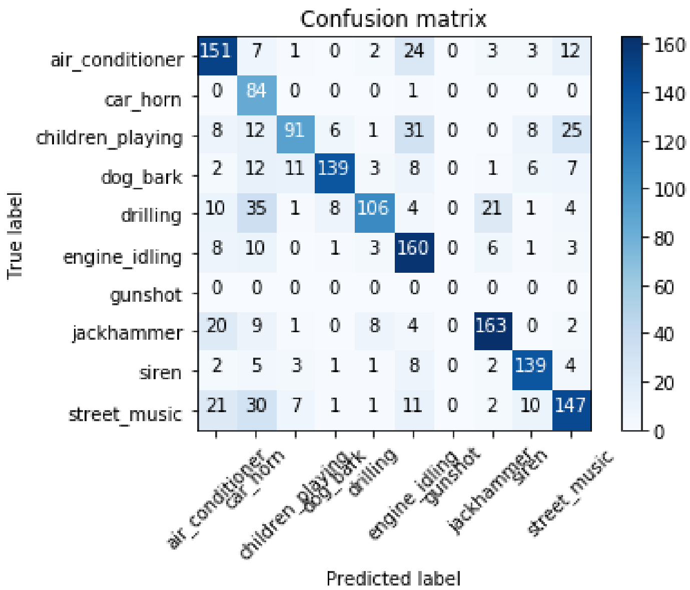


Fig. 5.1 Confusion Matrix

The Confusion Matrix tells a different story. Here we can see that our model struggles the most with the following sub-groups:

- ▪ air conditioner, jackhammer and street music.

        - car horn, drilling, and street music.
        - air conditioner, children playing and engine idling.
        - jackhammer and drilling.
    - air conditioner, car horn, children playing and street music.

This shows us that the problem is more nuanced than our initial assessment and gives some in-sights into the features that the CNN is extracting to make it's classifications. For example, street music is one of the commonly classified classes and could be to a wide variety of different samples within the class.

5.2. Justification

The ultimate show accomplished a classification exactness of 92% on the testing information which surpassed my desires given the benchmark was 68%.

Table 5.2.1 Classification Accuracy of Final and Benchmark Models

Model	Classification Accuracy
CNN	92%
MLP	88%
Benchmark SVM_rbf	68%

The ultimate arrangement performs well when displayed with a .wav record with a length of many seconds and returns a dependable classification.

However, we don't know how the demonstration would perform on Real-time sound. We don't know whether it would be able to perform the classification in an opportune manner so sound outlines are not skipped or the classification would be intensely influenced by latency.

Also, we don't know how the classifier would perform in a genuine world setting. Our think about makes no endeavor to decide the impact of components such as clamor, echos, volume, and striking nature level of the test.

5.3. Output Proof

Auditory inspection

We will use `IPython.display.Audio` to play the audio files so we can inspect aurally.

In [1]:
```
import IPython.display as ipd

ipd.Audio(r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold5\100032-3-0-0.wav")
```

Out[1]:

0:00 / 0:00

Visual inspection

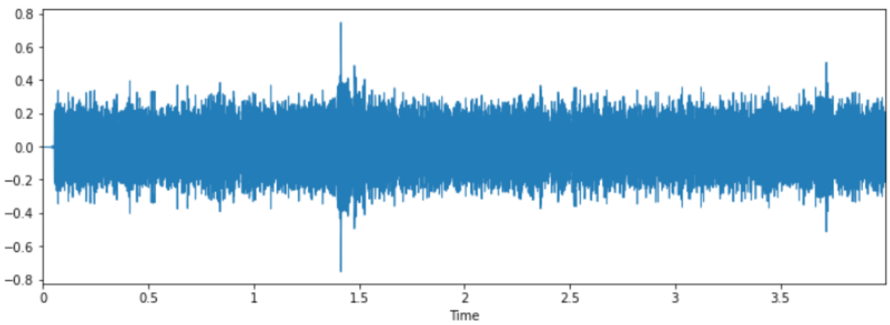
We will load a sample from each class and visually inspect the data for any patterns. We will use librosa to load the audio file into an array then librosa.display and matplotlib to display the waveform.

In [4]:
```
# Load imports

import IPython.display as ipd
import librosa
import librosa.display
import matplotlib.pyplot as plt
```

In [5]:
```
# Class: Air Conditioner

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold5\100852-0-0-0.wav")
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

Out[5]:

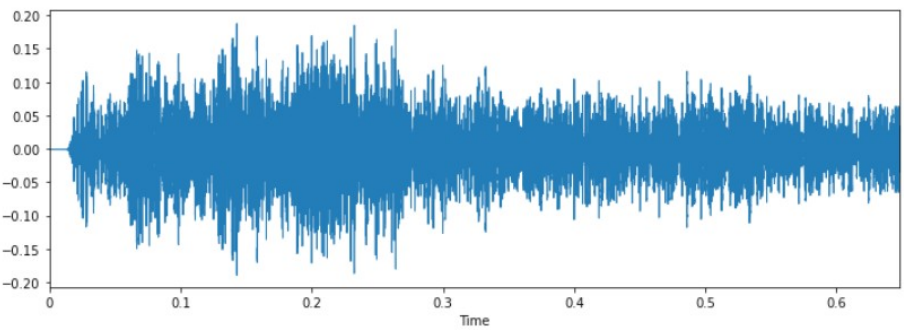
0:00 / 0:04



In [6]:
```
# Class: Car horn

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold10\100648-1-0-0.wav")
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

Out[6]:

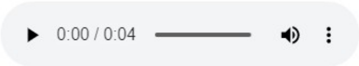
0:00 / 0:00

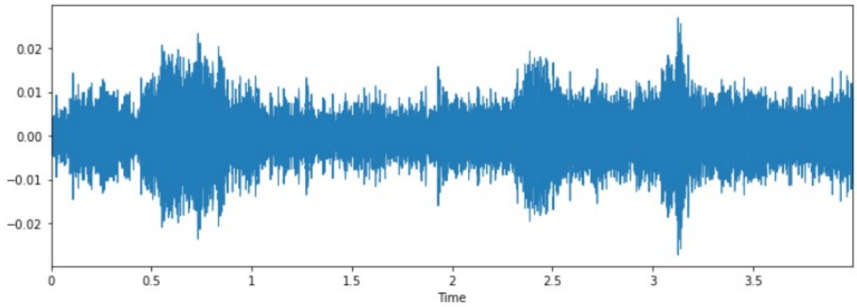


In [7]:

```python
# Class: Children playing

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold5\100263-2-0-117.wav")
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

Out[7]:

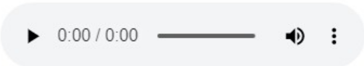


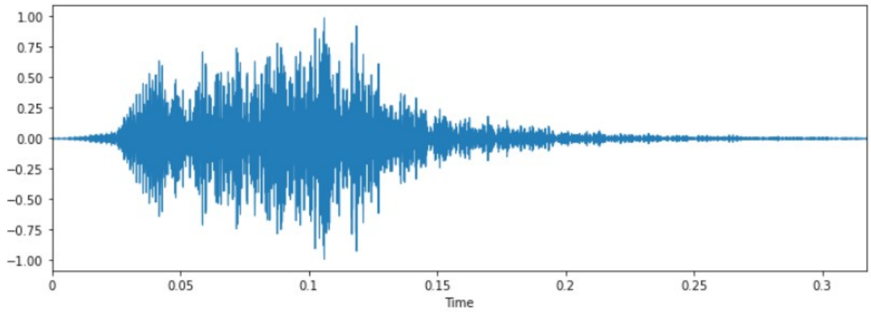


In [8]:

```python
# Class: Dog bark

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold5\100032-3-0-0.wav")
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

Out[8]:

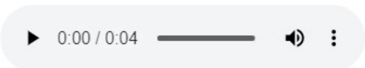


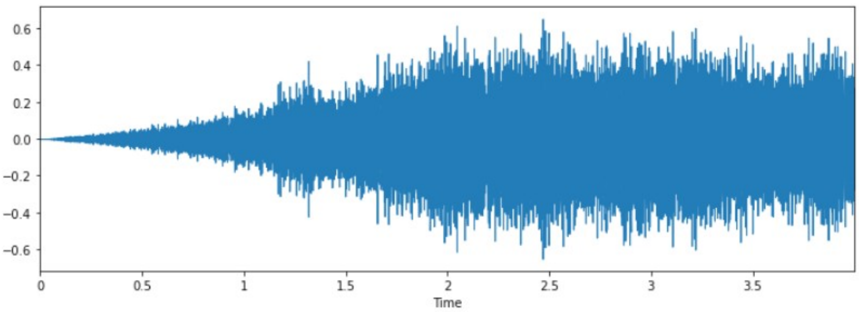


In [9]:

```python
# Class: Drilling

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold3\103199-4-0-0.wav")
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

Out[9]:



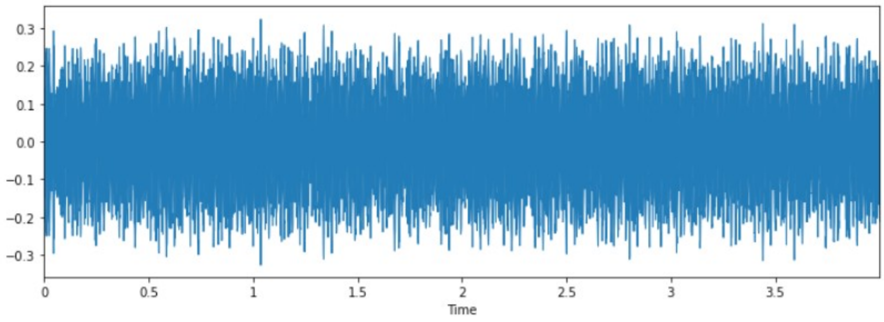


In [10]:

```python
# Class: Engine Idling

filename =(r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold10\102857-5-0-0.wav")
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

Out[10]:

0:00 / 0:04

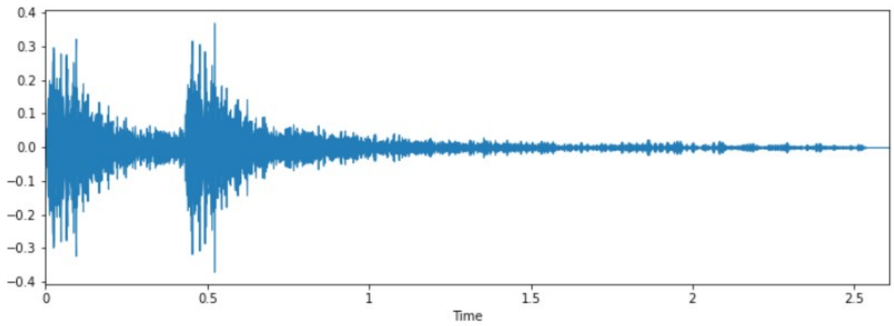


In [11]:

```python
# Class: Gunshot

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold1\102305-6-0-0.wav")
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

Out[11]:

0:00 / 0:02

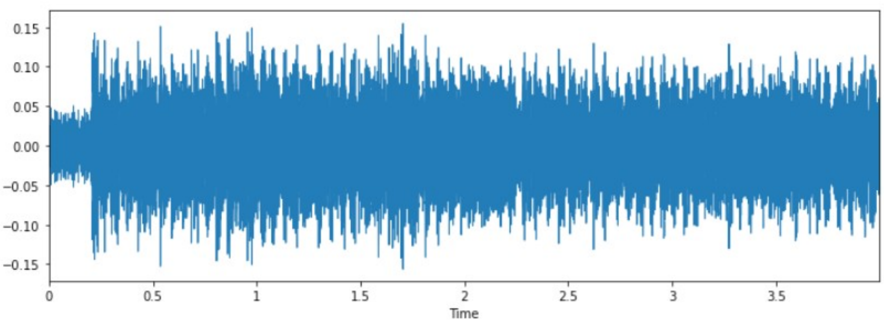


In [12]:

```python
# Class: Jackhammer

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold1\103074-7-0-0.wav")
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

Out[12]:

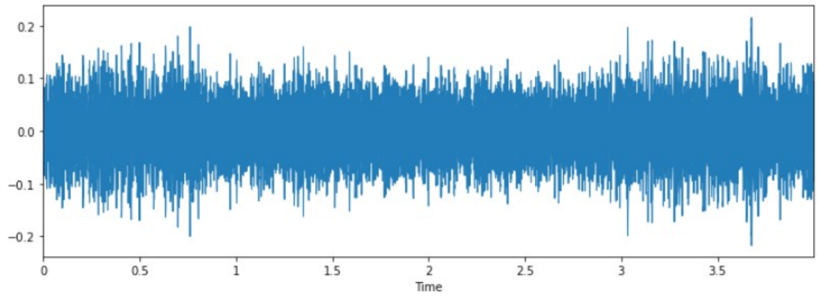
0:00 / 0:04



In [13]:
```python
# Class: Siren

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold7\102853-8-0-0.wav")
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

Out[13]:

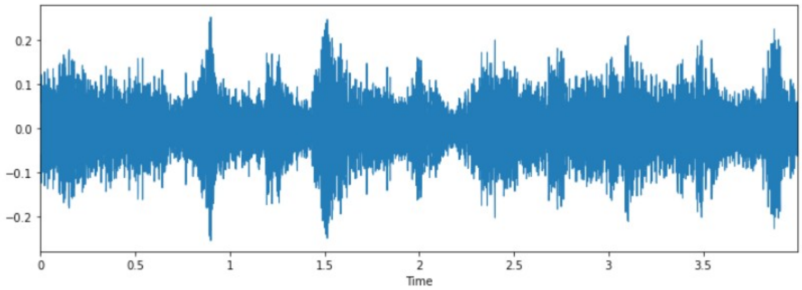
0:00 / 0:04



In [14]:
```python
# Class: Street music

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold7\101848-9-0-0.wav")
plt.figure(figsize=(12,4))
data,sample_rate = librosa.load(filename)
_ = librosa.display.waveplot(data,sr=sample_rate)
ipd.Audio(filename)
```

Out[14]:

0:00 / 0:04



Dataset Metadata

Here we will load the UrbanSound metadata .csv file into a Panda dataframe.

In [4]:
```python
import pandas as pd
import dataframe as df
metadata = pd.read_csv(r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\metadata\UrbanSound8K.csv")
metadata.head()
```

Out[4]:

	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.0	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.5	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.5	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.0	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.5	72.500000	1	5	2	children_playing

In [5]:
```python
type(metadata)
```

Out[5]: pandas.core.frame.DataFrame

Class distributions

In [6]:
```python
#print(metadata.class_name.value_counts())
```

Observations

Here we can see the Class labels are unbalanced. Although 7 out of the 10 classes all have exactly 1000 samples, and siren is not far off with 929, the remaining two (car_horn, gun_shot) have significantly less samples at 43% and 37% respectively.

This will be a concern and something we may need to address later on.

Sample rate

There is a wide range of Sample rates that have been used across all the samples which is a concern (ranging from 96k to 8k).

This likley means that we will have to apply a sample-rate conversion technique (either up-conversion or down-conversion) so we can see an agnostic representation of their waveform which will allow us to do a fair comparison.

In [18]:

```
# sample rates

print(audiodf.sample_rate.value_counts(normalize=True))
```

```
44100     0.614979
48000     0.286532
96000     0.069858
24000     0.009391
16000     0.005153
22050     0.005039
11025     0.004466
192000    0.001947
8000      0.001374
11024     0.000802
32000     0.000458
Name: sample_rate, dtype: float64
```

Bit-depth

There is also a wide range of bit-depths. It's likely that we may need to normalise them by taking the maximum and minimum amplitude values for a given bit-depth.

In [19]:

```
# bit depth

print(audiodf.bit_depth.value_counts(normalize=True))
```

```
16    0.659414
24    0.315277
32    0.019354
8     0.004924
4     0.001031
Name: bit_depth, dtype: float64
```

Audio sample file properties

Next we will iterate through each of the audio sample files and extract, number of audio channels, sample rate and bit-depth.

In [16]:

```
# Load various imports
import pandas as pd
import os
import librosa
import librosa.display

from helpers.wavfilehelper import WavFileHelper
wavfilehelper = WavFileHelper()

audiodata = []
for index, row in metadata.iterrows():

    file_name = os.path.join(os.path.abspath(r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train"),'
    data = wavfilehelper.read_file_properties(file_name)
    audiodata.append(data)

# Convert into a Panda dataframe
audiodf = pd.DataFrame(audiodata, columns=['num_channels','sample_rate','bit_depth'])
```

Audio channels

Most of the samples have two audio channels (meaning stereo) with a few with just the one channel (mono).

The easiest option here to make them uniform will be to merge the two channels in the stero samples into one by averaging the values of the two channels.

In [17]:

```
# num of channels

print(audiodf.num_channels.value_counts(normalize=True))
```

```
2    0.915369
1    0.084631
Name: num_channels, dtype: float64
```

Preprocessing stage

For much of the preprocessing we will be able to use [Librosa's load() function.]

We will compare the outputs from Librosa against the default outputs of [scipy's wavfile library] using a chosen file from the dataset.

Sample rate conversion

By default, Librosa's load function converts the sampling rate to 22.05 KHz which we can use as our comparison level.

In [20]:
```python
import librosa
from scipy.io import wavfile as wav
import numpy as np

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold5\100852-0-0-0.wav")

librosa_audio, librosa_sample_rate = librosa.load(filename)
scipy_sample_rate, scipy_audio = wav.read(filename)

print('Original sample rate:', scipy_sample_rate)
print('Librosa sample rate:', librosa_sample_rate)
```

```
Original sample rate: 44100
Librosa sample rate: 22050
```

Bit-depth

Librosa's load function will also normalise the data so it's values range between -1 and 1. This removes the complication of the dataset having a wide range of bit-depths.

In [21]:
```python
print('Original audio file min~max range:', np.min(scipy_audio), 'to', np.max(scipy_audio))
print('Librosa audio file min~max range:', np.min(librosa_audio), 'to', np.max(librosa_audio))
```

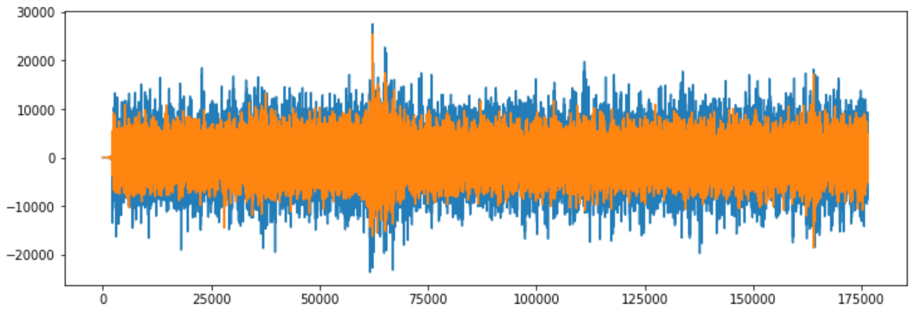
```
Original audio file min~max range: -23628 to 27507
Librosa audio file min~max range: -0.50266445 to 0.74983937
```

Merge audio channels

Librosa will also convert the signal to mono, meaning the number of channels will always be 1.

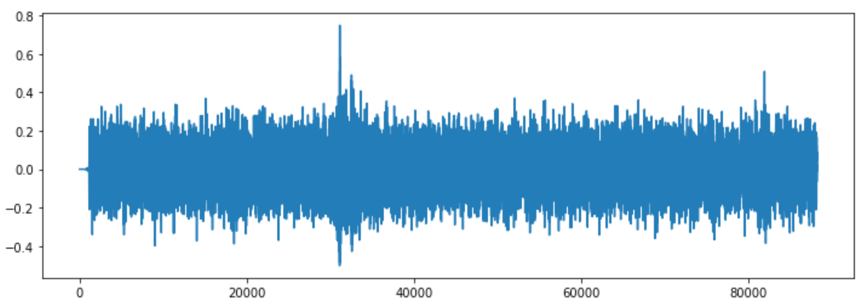
In [22]:
```python
import matplotlib.pyplot as plt

# Original audio with 2 channels
plt.figure(figsize=(12, 4))
plt.plot(scipy_audio)
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x227b9ac85f8>,
 <matplotlib.lines.Line2D at 0x227b9ac86a0>]
```



In [23]:
```python
# Librosa audio with channels merged
plt.figure(figsize=(12, 4))
plt.plot(librosa_audio)
```

Out[23]:
```
[<matplotlib.lines.Line2D at 0x227b9868d30>]
```



Extracting a MFCC

For this we will use Librosa's mfcc() function which generates an MFCC from time series audio data.

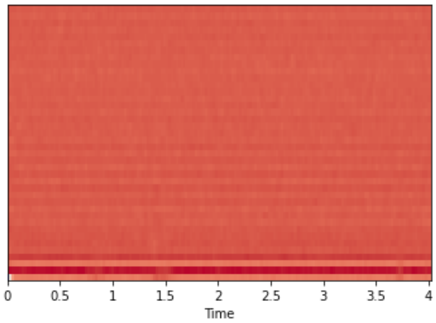
In [24]:
```
mfccs = librosa.feature.mfcc(y=librosa_audio, sr=librosa_sample_rate, n_mfcc=40)
print(mfccs.shape)
```

```
(40, 173)
```

This shows librosa calculated a series of 40 MFCCs over 173 frames.

In [25]:
```
import librosa.display
librosa.display.specshow(mfccs, sr=librosa_sample_rate, x_axis='time')
```

Out[25]: `<matplotlib.collections.QuadMesh at 0x227b9907f60>`



Extracting MFCC's for every file

We will now extract an MFCC for each audio file in the dataset and store it in a Panda Dataframe along with it's classification label.

In [7]:
```
def extract_features(file_name):

    try:
        audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
        mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
        mfccsscaled = np.mean(mfccs.T,axis=0)

    except Exception as e:
        print("Error encountered while parsing file: ", file)
        return None

    return mfccsscaled
```

In [15]:
```
# Load various imports
import pandas as pd
import os
import librosa

# Set the path to the full UrbanSound dataset
fulldatasetpath = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train")

metadata = pd.read_csv(r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\metadata\UrbanSound8K.csv")

features = []

# Iterate through each sound file and extract the features
for index, row in metadata.iterrows():

    file_name = os.path.join(os.path.abspath(r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train"),'

    class_label = row['class']
    data = extract_features(file_name)

    features.append([data, class_label])

# Convert into a Panda dataframe
featuresdf = pd.DataFrame(features, columns=['feature','class_label'])

print('Finished feature extraction from ', len(featuresdf), ' files')
```

```
C:\Users\Simriti Koul\Anaconda New\envs\Capstone\lib\site-packages\librosa\core\spectrum.py:224: UserWarning: n_fft=2048 is
too small for input signal of length=1323
  n_fft, y.shape[-1]
C:\Users\Simriti Koul\Anaconda New\envs\Capstone\lib\site-packages\librosa\core\spectrum.py:224: UserWarning: n_fft=2048 is
too small for input signal of length=1103
  n_fft, y.shape[-1]
C:\Users\Simriti Koul\Anaconda New\envs\Capstone\lib\site-packages\librosa\core\spectrum.py:224: UserWarning: n_fft=2048 is
too small for input signal of length=1523
  n_fft, y.shape[-1]
```

```
Finished feature extraction from  8732  files
```

Convert the data and labels

We will use `sklearn.preprocessing.LabelEncoder` to encode the categorical text data into model-understandable numerical data.

In [17]:
```
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

# Convert features and corresponding classification labels into numpy arrays
X = np.array(featuresdf.feature.tolist())
y = np.array(featuresdf.class_label.tolist())

# Encode the classification labels
le = LabelEncoder()
yy = to_categorical(le.fit_transform(y))
```

Split the dataset

Here we will use `sklearn.model_selection.train_test_split` to split the dataset into training and testing sets. The testing set size will be 20% and we will set a random state.

In [18]:
```
# split the dataset
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(X, yy, test_size=0.2, random_state = 42)
```

Store the preprocessed data

In [19]:
```
### store the preprocessed data for use in the next notebook

%store x_train
%store x_test
%store y_train
%store y_test
%store yy
%store le
```

```
Stored 'x_train' (ndarray)
Stored 'x_test' (ndarray)
Stored 'y_train' (ndarray)
Stored 'y_test' (ndarray)
Stored 'yy' (ndarray)
Stored 'le' (LabelEncoder)
```

Compiling the model

For compiling our model, we will use the following three parameters:

- Loss function - we will use `categorical_crossentropy`. This is the most common choice for classification. A lower score indicates that the model is performing better.
- Metrics - we will use the `accuracy` metric which will allow us to view the accuracy score on the validation data when we train the model.
- Optimizer - here we will use `adam` which is a generally good optimizer for many use cases.

In [5]:
```
# Compile the model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
```

In [*]:
```
# Display model architecture summary
model.summary()

# Calculate pre-training accuracy
score = model.evaluate(x_test, y_test, verbose=0)
accuracy = 100*score[1]

print("Pre-training accuracy: %.4f%%" % accuracy)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               10496
_________________________________________________________________
activation (Activation)      (None, 256)               0
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 78,858
Trainable params: 78,858
Non-trainable params: 0
```

Training

Here we will train the model.

We will start with 100 epochs which is the number of times the model will cycle through the data. The model will improve on each cycle until it reaches a certain point.

We will also start with a low batch size, as having a large batch size can reduce the generalisation ability of the model.

In [5]:

```
from keras.callbacks import ModelCheckpoint
from datetime import datetime

num_epochs = 100
num_batch_size = 32

checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.basic_mlp.hdf5',
                               verbose=1, save_best_only=True)
start = datetime.now()

model.fit(x_train, y_train, batch_size=num_batch_size, epochs=num_epochs, validation_data=(x_test, y_test), callbacks=[checkp


duration = datetime.now() - start
print("Training completed in time: ", duration)
```

```
Epoch 1/100
219/219 [==============================] - 4s 3ms/step - loss: 9.3813 - accuracy: 0.1984 - val_loss: 2.1696 - val_accura
cy: 0.2318

Epoch 00001: val_loss improved from inf to 2.16960, saving model to saved_models\weights.best.basic_mlp.hdf5
Epoch 2/100
219/219 [==============================] - 1s 3ms/step - loss: 2.2757 - accuracy: 0.2372 - val_loss: 2.0606 - val_accura
cy: 0.2604

Epoch 00002: val_loss improved from 2.16960 to 2.06056, saving model to saved_models\weights.best.basic_mlp.hdf5
Epoch 3/100
219/219 [==============================] - 1s 4ms/step - loss: 2.0418 - accuracy: 0.2911 - val_loss: 1.8194 - val_accura
cy: 0.3927

Epoch 00003: val_loss improved from 2.06056 to 1.81939, saving model to saved_models\weights.best.basic_mlp.hdf5
Epoch 4/100
219/219 [==============================] - 1s 3ms/step - loss: 1.8972 - accuracy: 0.3394 - val_loss: 1.6867 - val_accura
cy: 0.4367
```

Test the model

Here we will review the accuracy of the model on both the training and test data sets.

In [7]:

```
# Evaluating the model on the training and testing set
score = model.evaluate(x_train, y_train, verbose=0)
print("Training Accuracy: ", score[1])

score = model.evaluate(x_test, y_test, verbose=0)
print("Testing Accuracy: ", score[1])
```

```
Training Accuracy:  0.043664995580911636
Testing Accuracy:  0.041213508695364
```

The initial Training and Testing accuracy scores are quite high. As there is not a great difference between the Training and Test scores (~5%) this suggests that the model has not suffered from overfitting.

In [9]:

```
# Class: Air Conditioner

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold5\100852-0-0-0.wav")
print_prediction(filename)
```

```
C:\Users\Simriti Koul\Anaconda New\envs\Capstone\lib\site-packages\keras\engine\sequential.py:450: UserWarning: `model.predi
ct_classes()` is deprecated and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)
`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict
(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activatio
n).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

```
The predicted class is: air_conditioner

air_conditioner                  :  0.99765741825103759765625000000000
car_horn                 :  0.00051001412793993949890136718750
children_playing                 :  0.00010772443783935159444808959961
dog_bark                 :  0.00004354374686954542994499206543
drilling                 :  0.00042141648009419441223144531250
engine_idling            :  0.00103361869696527719497680664062
gun_shot                 :  0.00001595761568751186132431030273
jackhammer               :  0.00006289265729719772934913635254
siren            :  0.00000077757891858709626831114292
street_music             :  0.00014671032840851694345474243164
```

```
C:\Users\Simriti Koul\Anaconda New\envs\Capstone\lib\site-packages\keras\engine\sequential.py:425: UserWarning: `model.predi
ct_proba()` is deprecated and will be removed after 2021-01-01. Please use `model.predict()` instead.
  warnings.warn('`model.predict_proba()` is deprecated and '
```

In [18]:

```
# Class: Street music

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold7\101848-9-0-0.wav")
print_prediction(filename)
```

```
The predicted class is: street_music

air_conditioner                  :  0.00354291405528783798217773437500
car_horn                 :  0.00334771955385804176330566406250
children_playing                 :  0.02169414423406124114990234375000
dog_bark                 :  0.18936577439308166503906250000000
drilling                 :  0.00625380314886569976806640625000
engine_idling            :  0.00078949128510430455207824707031
gun_shot                 :  0.00902641285210847854614257812500
jackhammer               :  0.00607445463538169860839843750000
siren            :  0.00041220203274860978126525878906
street_music             :  0.75949299335479736328125000000000
```

In [17]:

```
# Class: Drilling

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold3\103199-4-0-0.wav")
print_prediction(filename)
```

```
The predicted class is: drilling

air_conditioner                  :  0.00000453069560535368509590625763
car_horn                 :  0.00013925565872341394424438476562
children_playing                 :  0.00147388770710676908493041992188
dog_bark                 :  0.00015168463869486004114151000977
drilling                 :  0.93388199806213378906250000000000
engine_idling            :  0.00000310904539446710259653627870
gun_shot                 :  0.00001169474489870481193065643311
jackhammer               :  0.00000526221401742077432572841644
siren            :  0.00000013799305520478810649365187
street_music             :  0.06433135271072387695312500000000
```

In [19]:

```
# Class: Car Horn

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold10\100648-1-0-0.wav")
print_prediction(filename)
```

```
The predicted class is: street_music

air_conditioner                  :  0.00472804810851812362670898437500
car_horn                 :  0.05722814798355102539062500000000
children_playing                 :  0.10663680732250213623046875000000
dog_bark                 :  0.19712772965431213378906250000000
drilling                 :  0.10537705570459365844726562500000
engine_idling            :  0.00423351302742958068847656250000
gun_shot                 :  0.04030582308769226074218750000000
jackhammer               :  0.13848775625228881835937500000000
siren            :  0.00154224026482552289962768554688
street_music             :  0.34433290362358093261718750000000
```

Other audio

Here we will use a sample of various copyright free sounds that we not part of either our test or training data to further validate our model.

In [13]:
```python
filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\Evaluation audio\dog_bark_1.wav")
print_prediction(filename)
```

```
The predicted class is: dog_bark

air_conditioner                 :  0.00030687192338518798351287841797
car_horn                :  0.00032225772156380116939544677734
children_playing                :  0.00474609946832060813903808593750
dog_bark                :  0.77336806058883666992187500000000
drilling                :  0.00888785626739263534545898437500
engine_idling           :  0.00018908230413217097520828247070
gun_shot                :  0.01179265696555376052856445312500
jackhammer              :  0.00010120595106855034828186035156
siren           :  0.00019697181414812803268432617188
street_music            :  0.20008899271488189697265625000000
```

In [14]:
```python
filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\Evaluation audio\drilling_1.wav")

print_prediction(filename)
```

```
The predicted class is: drilling

air_conditioner                 :  0.34108763933181762695312500000000
car_horn                :  0.00678393011912703514099121093750
children_playing                :  0.03632827475666999816894531250000
dog_bark                :  0.00861750636249780654907226562500
drilling                :  0.42608013749122619628906250000000
engine_idling           :  0.01366939768195152282714843750000
gun_shot                :  0.00453296350315213203430175781250
jackhammer              :  0.14550997316837310791015625000000
siren           :  0.00192271184641867876052856445312
street_music            :  0.01546743605285882949829101562500
```

In [15]:
```python
filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\Evaluation audio\gun_shot_1.wav")

print_prediction(filename)

# sample data weighted towards gun shot - peak in the dog barking sample is simmilar in shape to the gun shot sample
```

```
The predicted class is: dog_bark

air_conditioner                 :  0.06776662170886993408203125000000
car_horn                :  0.00015831571363378316164016723633
children_playing                :  0.00096621987177059054374694824219
dog_bark                :  0.55382043123245239257812500000000
drilling                :  0.00146359112113714218139648437500
engine_idling           :  0.00372890196740627288818359375000
gun_shot                :  0.00162448163609951734542846679688
jackhammer              :  0.00007820993778295814990997314453
siren           :  0.00039035922964103519916534423828
street_music            :  0.37000289559364318847656250000000
```

In [16]:
```python
filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\Evaluation audio\siren_1.wav")

print_prediction(filename)
```

```
The predicted class is: siren

air_conditioner                 :  0.00000010728005861437850398942828
car_horn                :  0.00000962881131272297352552413940
children_playing                :  0.00002930800656031351536512374878
dog_bark                :  0.01859312690794467926025390625000
drilling                :  0.00000646794342173961922526359558
engine_idling           :  0.02171797119081020355224609375000
gun_shot                :  0.00006894153921166434884071350098
jackhammer              :  0.00000164585208040080033242702484
siren           :  0.95715039968490600585937500000000
street_music            :  0.00242224242661148309707641601562 5
```

Observations

The performance of our initial model is satisfactorry and has generalised well, seeming to predict well when tested against new audio data.

In [3]:

```
# Load various imports
import pandas as pd
import os
import librosa

# Set the path to the full UrbanSound dataset
fulldatasetpath = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K")

metadata = pd.read_csv(r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\metadata\UrbanSound8K.csv")

features = []

# Iterate through each sound file and extract the features
for index, row in metadata.iterrows():

    file_name = os.path.join(os.path.abspath(r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train"),'

    class_label = row["class"]
    data = extract_features(file_name)

    features.append([data, class_label])

# Convert into a Panda dataframe
featuresdf = pd.DataFrame(features, columns=['feature','class_label'])

print('Finished feature extraction from ', len(featuresdf), ' files')
```

```
C:\Users\Simriti Koul\Anaconda New\envs\Capstone\lib\site-packages\librosa\core\spectrum.py:224: UserWarning: n_fft=2048 is
too small for input signal of length=1323
  n_fft, y.shape[-1]
C:\Users\Simriti Koul\Anaconda New\envs\Capstone\lib\site-packages\librosa\core\spectrum.py:224: UserWarning: n_fft=2048 is
too small for input signal of length=1103
  n_fft, y.shape[-1]
C:\Users\Simriti Koul\Anaconda New\envs\Capstone\lib\site-packages\librosa\core\spectrum.py:224: UserWarning: n_fft=2048 is
too small for input signal of length=1523
  n_fft, y.shape[-1]
```

```
Finished feature extraction from  8732  files
```

Compiling the model

For compiling our model, we will use the same three parameters as the previous model:

In [9]:

```
# Compile the model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
```

In [10]:

```
# Display model architecture summary
model.summary()

# Calculate pre-training accuracy
score = model.evaluate(x_test, y_test, verbose=1)
accuracy = 100*score[1]

print("Pre-training accuracy: %.4f%%" % accuracy)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 39, 173, 16)       80
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 19, 86, 16)        0
_________________________________________________________________
dropout (Dropout)            (None, 19, 86, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 18, 85, 32)        2080
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 9, 42, 32)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 9, 42, 32)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 41, 64)         8256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 20, 64)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 20, 64)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 19, 128)        32896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 9, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 1, 9, 128)         0
_________________________________________________________________
global_average_pooling2d (Gl (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 10)                1290
=================================================================
Total params: 44,602
Trainable params: 44,602
Non-trainable params: 0
_________________________________________________________________
55/55 [==============================] - 24s 35ms/step - loss: 9.0762 - accuracy: 0.0380
Pre-training accuracy: 3.6062%
```

In [11]:

```
from keras.callbacks import ModelCheckpoint
from datetime import datetime

#num_epochs = 12
#num_batch_size = 128

num_epochs = 72
num_batch_size = 256

checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.basic_cnn.hdf5',
                               verbose=1, save_best_only=True)
start = datetime.now()

model.fit(x_train, y_train, batch_size=num_batch_size, epochs=num_epochs, validation_data=(x_test, y_test), callbacks=[checkp


duration = datetime.now() - start
print("Training completed in time: ", duration)
```

```
Epoch 1/72
28/28 [==============================] - 28s 865ms/step - loss: 5.6707 - accuracy: 0.1591 - val_loss: 2.1975 - val_accura
cy: 0.2129

Epoch 00001: val_loss improved from inf to 2.19753, saving model to saved_models\weights.best.basic_cnn.hdf5
Epoch 2/72
28/28 [==============================] - 25s 894ms/step - loss: 2.1457 - accuracy: 0.2744 - val_loss: 2.0202 - val_accura
cy: 0.2736

Epoch 00002: val_loss improved from 2.19753 to 2.02024, saving model to saved_models\weights.best.basic_cnn.hdf5
Epoch 3/72
28/28 [==============================] - 26s 919ms/step - loss: 1.7939 - accuracy: 0.3591 - val_loss: 1.8236 - val_accura
cy: 0.3887

Epoch 00003: val_loss improved from 2.02024 to 1.82358, saving model to saved_models\weights.best.basic_cnn.hdf5
Epoch 4/72
28/28 [==============================] - 26s 944ms/step - loss: 1.6290 - accuracy: 0.4222 - val_loss: 1.6935 - val_accura
cy: 0.4345

Epoch 00004: val_loss improved from 1.82358 to 1.69347, saving model to saved_models\weights.best.basic_cnn.hdf5
```

Test the model

Here we will review the accuracy of the model on both the training and test data sets.

In [12]:

```
# Evaluating the model on the training and testing set
score = model.evaluate(x_train, y_train, verbose=0)
print("Training Accuracy: ", score[1])

score = model.evaluate(x_test, y_test, verbose=0)
print("Testing Accuracy: ", score[1])
```

```
Training Accuracy:  0.9341446161270142
Testing Accuracy:  0.8712077736854553
```

The Training and Testing accuracy scores are both high and an increase on our initial model. Training accuracy has increased by ~6% and Testing accuracy has increased by ~4%.

There is a marginal increase in the difference between the Training and Test scores (~6% compared to ~5% previously) though the difference remains low so the model has not suffered from overfitting.

Validation

Test with sample data

As before we will verify the predictions using a subsection of the sample audio files we explored in the first notebook. We expect the bulk of these to be classified correctly.

In [14]:

```
# Class: Air Conditioner

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold5\100852-0-0-0.wav")
print_prediction(filename)
```

```
C:\Users\Simriti Koul\Anaconda New\envs\Capstone\lib\site-packages\keras\engine\sequential.py:450: UserWarning: `model.predi
ct_classes()` is deprecated and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)
`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict
(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activatio
n).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

```
The predicted class is: air_conditioner

air_conditioner                 :  0.98654663562774658203125000000000
car_horn                :  0.00000365796358892112039029598236
children_playing                :  0.00373674952425062656402587890625
dog_bark                :  0.00031396126723848283290863037109
drilling                :  0.00213159387931227684020996093750
engine_idling           :  0.00406803796067833900451660156250
gun_shot                :  0.00000933662704483140259981155396
jackhammer              :  0.00316123594529926776885986328125
siren           :  0.00000129648049096431350335478783
street_music            :  0.00002752682303253095597028732300
```

```
C:\Users\Simriti Koul\Anaconda New\envs\Capstone\lib\site-packages\keras\engine\sequential.py:425: UserWarning: `model.predi
ct_proba()` is deprecated and will be removed after 2021-01-01. Please use `model.predict()` instead.
  warnings.warn('`model.predict_proba()` is deprecated and '
```

In [15]:

```python
# Class: Drilling

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold3\103199-4-0-0.wav")
print_prediction(filename)
```

```
The predicted class is: drilling

air_conditioner                 :  0.00000014084022836868825834244490
car_horn                :  0.02140823565423488616943359375000
children_playing                :  0.00003284804915892891585826873779
dog_bark                :  0.00000577101127419155091047286987
drilling                :  0.88445901870727539062500000000000
engine_idling           :  0.00000012620598965895624132826924
gun_shot                :  0.00000104415573787264293059706688
jackhammer              :  0.00000748574166209436953067779541
siren           :  0.00000002659186293385573662817478
street_music            :  0.09408541023731231689453125000000
```

In [16]:

```python
# Class: Street music

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold7\101848-9-0-0.wav")
print_prediction(filename)
```

```
The predicted class is: street_music

air_conditioner                 :  0.00064984644996002316474914550781
car_horn                :  0.00292213051579892635345458984375
children_playing                :  0.02916814200580120086669921875000
dog_bark                :  0.00079743505921214818954467773438
drilling                :  0.00000521766332894912920892238617
engine_idling           :  0.00000295795189231284894049167633
gun_shot                :  0.00000000020001218736798165309665
jackhammer              :  0.00000089540787939768051728606224
siren           :  0.01239700336009263992309570312500
street_music            :  0.95405632257461547851562500000000
```

In [17]:

```python
# Class: Car Horn

filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\UrbanSound8K.tar\UrbanSound8K\train\fold10\100648-1-0-0.wav")
print_prediction(filename)
```

```
The predicted class is: car_horn

air_conditioner                 :  0.00219908985309302806854248046875
car_horn                :  0.30767381191253662109375000000000
children_playing                :  0.00892277713865041732788085937500
dog_bark                :  0.09442199766635894775390625000000
drilling                :  0.25490027666091918945312500000000
engine_idling           :  0.01859729923307895660400390625000
gun_shot                :  0.07564551383256912231445312500000
jackhammer              :  0.20189863443374633789062500000000
siren           :  0.03126519173383712768554687500000
street_music            :  0.00447552790865302085876464843750
```

Observations

We can see that the model performs well.

Interestingly, car horn was again incorrectly classifed but this time as drilling - though the per class confidence shows it was a close decision between car horn with 26% confidence and drilling at 34% confidence.

In [19]:

```python
filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\Evaluation audio\drilling_1.wav")

print_prediction(filename)
```

```
The predicted class is: jackhammer

air_conditioner                 :  0.01153741963207721710205078125000
car_horn                :  0.00212912564165890216827392578125
children_playing                :  0.00002556872641434893012046813965
dog_bark                :  0.00049320107791572809219360351562
drilling                :  0.08964597433805465698242187500000
engine_idling           :  0.00117059098556637763977050781250
gun_shot                :  0.00000086370192775575560517609119
jackhammer              :  0.89416062831878662109375000000000
siren           :  0.00076230237027630209922790527344
street_music            :  0.00007435309089487418532371520996
```

Other audio

Again we will further validate our model using a sample of various copyright free sounds that we not part of either our test or training data.

In [18]:

```
filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\Evaluation audio\dog_bark_1.wav")
print_prediction(filename)
```

```
The predicted class is: dog_bark

air_conditioner                  :  0.00169448740780353546142578125000
car_horn                 :  0.02364376187324523925781250000000
children_playing                 :  0.00123100646305829286575317382812
dog_bark                 :  0.88138276338577270507812500000000
drilling                 :  0.04747003316879272460937500000000
engine_idling            :  0.00059739599237218499183654785156
gun_shot                 :  0.03793414682149887084960937500000
jackhammer               :  0.00192393758334219455718994140625
siren            :  0.00134987547062337398529052734375
street_music             :  0.00277261622250080108642578125000
```

In [20]:

```
filename = (r"C:\Users\Simriti Koul\Desktop\CAPSTONE\Evaluation audio\gun_shot_1.wav")

print_prediction(filename)
```

```
The predicted class is: gun_shot

air_conditioner                  :  0.00051879772217944264411926269531
car_horn                 :  0.00005635288107441738247871398926
children_playing                 :  0.00134165945928543806076049804688
dog_bark                 :  0.15568615496158599853515625000000
drilling                 :  0.01065753120929002761840820312500
engine_idling            :  0.00056926830438897013664245605469
gun_shot                 :  0.83017569780349731445312500000000
jackhammer               :  0.00000720739217285881750285625458
siren            :  0.00025452961563132703304290771484
street_music             :  0.00073273148154839873313903808594
```

Observations

The performance of our final model is very good and has generalised well, seeming to predict well when tested against new audio data.

6. REFERENCES

[1] Justin Salamon, Christopher Jacoby and Juan Pablo Bello. Urban Sound Datasets.

https://urbansounddataset.weebly.com/urbansound8k.html

[2] Mel-frequency cepstrum Wikipedia page

https://en.wikipedia.org/wiki/Mel-frequency_cepstrum

[3] J. Salamon, C. Jacoby, and J. P. Bello. A dataset and taxonomy for urban sound research.

http://www.justinsalamon.com/uploads/4/3/9/4/4394963/salamon_urbansound_acmmm14.pdf

[4] Manik Soni AI which classifies Sounds:Code:Python.

https://hackernoon.com/ai-which-classifies-sounds-code-python-6a07a204381032

[5] Manash Kumar Mandal Building a Dead Simple Speech Recognition Engine using ConvNet in Keras. https://blog.manash.me/building-a-dead-simple-word-recognition-engine-using-convnet-in-keras-25e72c19c12b

[6] Eijaz Allibhai Building a Convolutional Neural Network (CNN) in Keras.

https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5

[7] Daphne Cornelisse An intuitive guide to Convolutional Neural Networks.

https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

[8] Urban Sound Classification - Part 2: sample rate conversion, Librosa.

https://towardsdatascience.com/urban-sound-classification-part-2-sample-rate-conversion-librosa-ba7bc88f209a

[9] wavsource.com THE Source for Free Sound Files and Reviews.

http://www.wavsource.com/

[10] soundbible.com. http://soundbible.com//

[11] D. Bogdanov, N. Wack, E. G´omez, S. Gulati, P. Herrera, O. Mayor, G. Roma, J. Salamon, J. Zapata, and X. Serra. ESSENTIA: an open-source library for sound and music analysis. In 21st ACM Int. Conf. on Multimedia, pages 855–858, 2013.

[12] A. L. Brown, J. Kang, and T. Gjestland. Towards standardization in soundscape preference assessment. Applied Acoustics, 72(6):387–392, 2011.

[13] L.-H. Cai, L. Lu, A. Hanjalic, H.-J. Zhang, and L.-H. Cai. A flexible framework for key audio effects detection and auditory context inference. IEEE TASLP, 14(3):1026–1039, 2006.

[14] S. Chaudhuri and B. Raj. Unsupervised hierarchical structure induction for deeper semantic analysis of audio. In IEEE ICASSP, pages 833–837, 2013.

[15] S. Chu, S. Narayanan, and C.-C. Kuo. Environmental sound recognition with time-frequency audio features. IEEE TASLP, 17(6):1142–1158, 2009.

[16] C. V. Cotton and D. P. W. Ellis. Spectral vs. spectro-temporal features for acoustic event detection. In IEEE WASPAA'11, pages 69–72, 2011.

[17] D. P. W. Ellis, X. Zeng, and J. H. McDermott. Classifying soundtracks with audio texture features. In IEEE ICASSP, pages 5880–5883, 2011.

[18] D. Giannoulis, D. Stowell, E. Benetos, M. Rossignol, M. Lagrange, and M. D. Plumbley. A database and challenge for acoustic scene classification and event detection. In 21st EUSIPCO, 2013.

[19] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. SIGKDD Explorations Newsletter, 11(1):10–18, 2009.

[20] T. Heittola, A. Mesaros, A. Eronen, and T. Virtanen. Audio context recognition using audio event histograms. In 18th EUSIPCO, pages 1272–1276, 2010.

[21] T. Heittola, A. Mesaros, A. Eronen, and T. Virtanen. Context-dependent sound event detection. EURASIP JASMP, 2013(1), 2013.

[22] S. R. Payne, W. J. Davies, and M. D. Adams. Research into the practical and policy applications of soundscape concepts and techniques in urban areas. DEFRA, HMSO, London, UK, 2009.

[23] R. Radhakrishnan, A. Divakaran, and P. Smaragdis. Audio analysis for surveillance applications. In IEEE WASPAA'05, pages 158–161, 2005.

[24] R. M. Schafer. The Soundscape: Our Sonic Environment and the Tuning of the World. Destiny Books, 1993.

[25] D. Steele, J. D. Krijnders, and C. Guastavino. The sensor city initiative: cognitive sensors for soundscape transformations. In GIS Ostrava, pages 1–8, 2013.

[26] M. Xu, C. Xu, L. Duan, J. S. Jin, and S. Luo. Audio keywords generation for sports video analysis. ACM TOMCCAP, 4(2):1–23, 2008

[27] S Ntalampiras, I Potamitis, N Fakotakis, in IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '09. On acoustic surveillance of hazardous situations (IEEE Computer Society, Washington, DC, USA, 2009), pp. 165–168

[28] S Chu, S Narayanan, CCJ Kuo, Environmental sound recognition with time-frequency audio features. IEEE Trans. Audio Speech Lang. Process. 17(6), 1142–1158 (2009)

[29] T Heittola, A Mesaros, A Eronen, T Virtanen, in 18th European Signal Processing Conference. Audio context recognition using audio event histograms (Aalborg, Denmark, 2010), pp. 1272–1276

[30] M Shah, B Mears, C Chakrabarti, A Spanias, in 2012 IEEE International Conference on Emerging Signal Processing Applications (ESPA). Lifelogging: archival and retrieval of continuously recorded audio using wearable

devices (IEEE Computer Society, Las Vegas, NV, USA, 2012), pp. 99–102

[31] G Wichern, J Xue, H Thornburg, B Mechtley, A Spanias, Segmentation, indexing, and retrieval for environmental and natural sounds. IEEE Trans. Audio Speech Lang. Process. 18(3), 688–707 (2010)

[32] AS Bregman, Auditory Scene Analysis. (MIT Press, Cambridge MA, 1990)

[33] M Bar, The proactive brain: using analogies and associations to generate predictions. Trends Cogn. Sci. 11(7), 280–289 (2007)

[34] A Oliva, A Torralba, The role of context in object recognition. Trends Cogn. Sci.

11(12), 520–527 (2007)

[35] M Niessen, L van Maanen, T Andringa, in IEEE International Conference on Semantic Computing. Disambiguating sounds through context (IEEE Computer Society, Santa Clara, CA, USA, 2008), pp. 88–95

[36] C Clavel, T Ehrette, G Richard, in IEEE International Conference on Multimedia and Expo. Events detection for an audio-based surveillance system (IEEE Computer Society, Los Alamitos, CA, USA, 2005), pp. 1306–1309

[37] H Wu, J Mendel, Classification of battlefield ground vehicles using acoustic features and fuzzy logic rule-based classifiers. IEEE Trans. Fuzzy Syst. 15, 56–72 (2007)

[38] L Atlas, G Bernard, S Narayanan, Applications of time-frequency analysis to signals from manufacturing and machine monitoring sensors. Proc. IEEE. 84(9), 1319–1329 (1996)

[39] S Fagerlund, Bird species recognition using support vector machines. EURASIP J. Appl. Signal Process. 2007, 64–64 (2007)

[40] F Kraft, R Malkin, T Schaaf, A Waibel, in Proceedings of Interspeech. Temporal ICA for classification of acoustic events in a kitchen environment (International Speech Communication Association, Lisboa, Portugal, 2005), pp. 2689–2692

[41] J Chen, AH Kam, J Zhang, N Liu, L Shue, in Pervasive Computing. Bathroom activity monitoring based on sound (Springer, Berlin, 2005), pp. 47–61

[42] A Temko, C Nadeu, Classification of acoustic events using SVM-based clustering schemes. Pattern Recognit. 39(4), 682–694 (2006)

[43] TH Dat, H Li, in IEEE International Conference on Acoustics, Speech and Signal Processing. Probabilistic distance SVM with Hellinger-exponential kernel for sound event classification (IEEE Computer Society, Prague, Czech Republic, 2011), pp. 2272–2275

[44] R Stiefelhagen, R Bowers, J(eds) Fiscus, Multimodal Technologies for Perception of Humans: International Evaluation Workshops CLEAR 2007 and RT 2007. (Springer, Berlin Germany, 2008)

[45] X Zhou, X Zhuang, M Liu, H Tang, M Hasegawa-Johnson, T Huang, in

Multimodal Technologies for Perception of Humans: International Evaluation Workshops CLEAR 2007 and RT 2007. HMM-based acoustic event detection with AdaBoost feature selection (Springer, Berlin, Germany, 2008), pp. 345–353

[46] X Zhuang, X Zhou, MA Hasegawa-Johnson, TS Huang, Real-world acoustic event detection. Pattern Recognit. Lett. (Pattern Recognition of Non-Speech Audio). 31(12), 1543–1551 (2010)

[47] A Mesaros, T Heittola, A Eronen, T Virtanen, in 18th European Signal Processing Conference. Acoustic event detection in real-life recordings (Aalborg, Denmark, 2010), pp. 1267–1271

[48] M Akbacak, JHL Hansen, Environmental sniffing: noise knowledge estimation for robust speech systems. IEEE Trans. Audio Speech Lang. Process. 15(2), 465–477 (2007)

[49] A Mesaros, H Heittola, A Klapuri, in 19th European Signal Processing Conference. Latent semantic analysis in sound event detection (Barcelona, Spain, 2011), pp. 1307-1311

[50] A Eronen, V Peltonen, J Tuomi, A Klapuri, S Fagerlund, T Sorsa, G Lorho, J Huopaniemi, Audio-based context recognition. IEEE Trans. Audio Speech Lang. Process. 14, 321–329 (2006)

[51] JJ Aucouturier, B Defreville, F Pacher, The bag-of-frames approach to audio pattern recognition: a sufficient model for urban soundscapes but not for polyphonic music. J. Acoust. Soc. Am. 122(2), 881–891 (2007)

[52] R Cai, L Lu, A Hanjalic, Co-clustering for auditory scene categorization. IEEE Trans. Multimed. 10(4), 596–606 (2008)

[53] L Lie, A Hanjalic, Text-like segmentation of general audio for content-based retrieval. IEEE Trans. Multimed. 11(4), 658–669 (2009)

[54] LR Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE. 77(2), 257–286 (1989)

[55] D Reynolds, R Rose, Robust text-independent speaker identification using

Gaussian mixture speaker models. IEEE Trans. Speech Audio Process. 3, 72–83 (1995)

[56] GD Forney, The Viterbi algorithm. Proc. IEEE. 61(3), 268–278 (1973)

[57] M Ryynanen, A Klapuri, in ¨ Proceedings of the 2005 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. Polyphonic music transcription using note event modeling (IEEE Computer Society, New York, NY, USA, 2005), pp. 319–322

[58] A Temko, C Nadeu, D Macho, R Malkin, C Zieger, M Omologo, in Computers in the Human Interaction Loop, ed. by AH Waibel, R Stiefelhagen. Acoustic event detection and classification (Springer, New York, 2009), pp. 61–73

[59] M Grootel, T Andringa, J Krijnders, in Proceedings of the NAG/DAGA Meeting 2009. DARES-G1: database of annotated real-world everyday sounds (Rotterdam, Netherlands, 2009), pp. 996–999

[60] T Heittola, A Mesaros, T Virtanen, A Eronen, in Workshop on Machine Listening in Multisource Environments, CHiME2011. Sound event detection in multisource environments using source separation (Florence, Italy, 2011), pp. 36–40

[61] S. Chu, S. Narayanan, and C.-C.J. Kuo, "Environmental sound recognition with time-frequency audio features," IEEE TASLP, vol. 17, no. 6, pp. 1142–1158, 2009.

[62] R. Radhakrishnan, A. Divakaran, and P. Smaragdis, "Audio analysis for surveillance applications," in IEEE WASPAA'05, 2005, pp. 158–161.

[63] M. Xu, C. Xu, L. Duan, J. S. Jin, and S. Luo, "Audio keywords generation for sports video analysis," ACM TOMCCAP, vol. 4, no. 2, pp. 1–23, 2008.

[64] Y.-F. Ma, L. Lu, H.-J. Zhang, and M. Li, "A user attention model for video summarization," in 10th ACM Int. Conf. On Multimedia, 2002, pp. 533–542.

[65] D. Steele, J. D. Krijnders, and C. Guastavino, "The sensor city initiative: cognitive sensors for soundscape transformations," in GIS Ostrava, 2013, pp. 1–8.

[66] D. P. W. Ellis and K. Lee, "Minimal-impact audio-based personal archives," in 1st ACM workshop on Continuous archival and retrieval of personal experiences,

New York, NY, USA, Oct. 2004, pp. 39–47.

[67] T. Heittola, A. Mesaros, A. Eronen, and T. Virtanen, “Audio context recognition using audio event histograms,” in 18th EUSIPCO, 2010, pp. 1272–1276.

[68] S. Chaudhuri and B. Raj, “Unsupervised hierarchical structure induction for deeper semantic analysis of audio,” in IEEE ICASSP, 2013, pp. 833–837.

[69] L.-H. Cai, L. Lu, A. Hanjalic, H.-J. Zhang, and L.-H Cai, “A flexible framework for key audio effects detection and auditory context inference,” IEEE TASLP, vol. 14, no. 3, pp. 1026–1039, 2006.

[70] T. Heittola, A. Mesaros, A. Eronen, and T. Virtanen, “Context dependent sound event detection,” EURASIP JASMP, vol. 2013, no. 1, 2013.

[71] T. Ganchev, N. Fakotakis, and G. Kokkinakis, “Comparative evaluation of various MFCC implementations on the speaker verification task,” in 10th Int. Conf. on Speech and Computer, Patras, Greece, Oct. 2005, vol. 1, pp. 191–194.

[72] C. V. Cotton and D. P. W. Ellis, “Spectral vs. spectro-temporal features for acoustic event detection,” in IEEE WASPAA’11, 2011, pp. 69–72.

[73] D. Stowell and M. D. Plumbley, “Automatic large-scale classification of bird sounds is strongly improved by unsupervised feature learning,” PeerJ, vol. 2, pp. e488, Jul. 2014.

[74] S. Dieleman and B. Schrauwen, “Multiscale approaches to music audio feature learning,” in 14th Int. Soc. for Music Info. Retrieval Conf., Curitiba, Brazil, Nov. 2013.

[75] P. Hamel, S. Lemieux, Y. Bengio, and D. Eck, “Temporal pooling and multiscale learning for automatic annotation and ranking of music audio.,” in 12th Int. Soc. for Music Info. Retrieval Conf., Miami, USA, Oct. 2011, pp. 729–734.

[76] Y. Vaizman, B. McFee, and G. Lanckriet, “Codebook-based audio feature representation for music information retrieval,” IEEE Transactions on Audio, Speech, and Language Processing, vol. 22, no. 10, pp. 1483–1493, Oct. 2014.

[77] E. Amid, A. Mesaros, K. J Palomaki, J. Laaksonen, and M. Kurimo, “Unsupervised feature extraction for multimedia event detection and ranking using audio content,” in IEEE Int. Conf. on Acoustics, Speech and Signal

Processing (ICASSP), Florence, Italy, May 2014, pp. 5939–5943.

[78] A. Coates and A. Y. Ng, “Learning feature representations with K-means,” in Neural Networks: Tricks of the Trade, pp. 561–580. Springer, 2012.

[79] J. Salamon, C. Jacoby, and J. P. Bello, “A dataset and taxonomy for urban sound research,” in 22nd ACM International Conference on Multimedia (ACM-MM’14), Orlando, FL, USA, Nov. 2014.

[80] D. Bogdanov, N. Wack, E. Gomez, S. Gulati, P. Herrera, O. Mayor, G. Roma, J. Salamon, J. Zapata, and X. Serra, “ESSENTIA: an audio analysis library for music information retrieval,” in 14th Int. Soc. for Music Info. Retrieval Conf., Curitiba, Brazil, Nov. 2013, pp. 493–498.

[81] S. Lloyd, “Least squares quantization in PCM,” IEEE Trans. on Information Theory, vol. 28, no. 2, pp. 129–137, 1982.

[82] I.S. Dhillon and D.M. Modha, “Concept decompositions for large sparse text data using clustering,” Machine Learning, vol. 42, no. 1, pp. 143–175, 2001.

[83] L. Breiman, “Random forests,” Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.

[84] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in python,” J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

[85] T. Fawcett, “An introduction to ROC analysis,” Pattern Recognition Letters, vol. 27, no. 8, pp. 861–874, 2006.

[86] D. P. W. Ellis, X. Zeng, and J. H. McDermott, “Classifying soundtracks with audio texture features,” in IEEE ICASSP, 2011, pp. 5880–5883