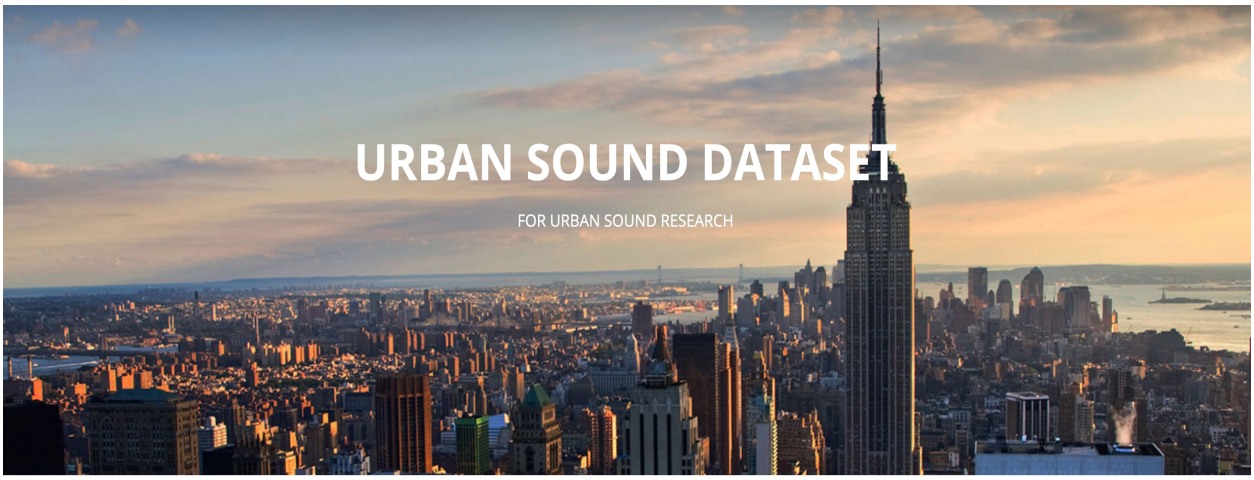
M11: Final Report



URBAN SOUND CLASSIFICATION

Dimitris Sideris | May 2019
Big Data Class M11

Abstract

In this project report we will present some basics about sound analysis that are relevant to our classification approach, the nature of our selected dataset and interesting findings from existing literature.

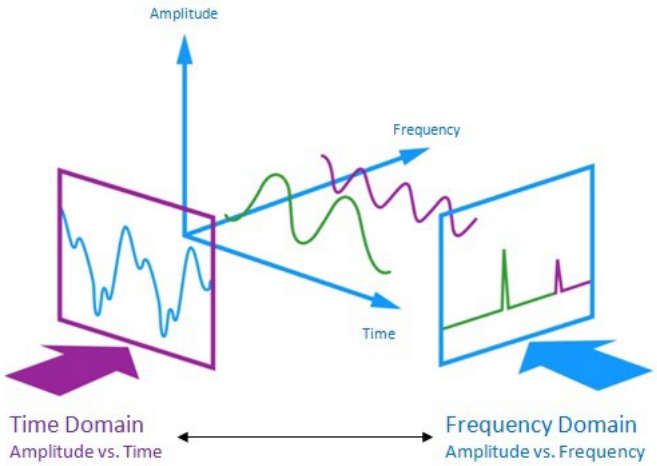
Environmental/Urban sound classification is a new subject and not much research is done comparing to music genre classification not to mention image classification. However there are some interesting papers that are dealing with urban sound analysis and classification that helped us navigate to this new upcoming topic. [3][4][5]

It seems that the most popular and performant solution of urban sound classification is to extract different kind of features from raw audio data into images and then perform image classification. Given this main idea some interesting proposed solutions exist in literature that try to experiment with combinations of different approaches.

From our research sound classification through image classification using convolutional neural networks seems to be the most efficient approach. Using MEL spectrograms, MFCC spectrograms, Chromagrams and Convolutional Neural Networks will be our main approach of implementing our urban sound classification.

1. Sound Basics for our project

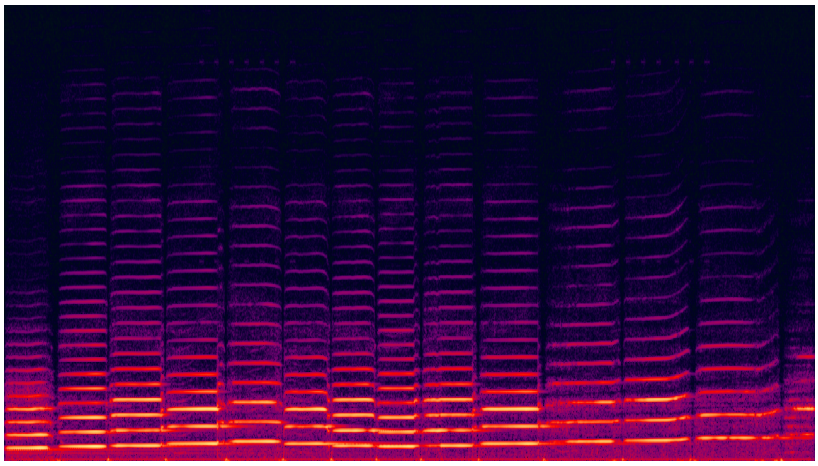
The audio signal is a three-dimensional signal in which three axes represent time, amplitude and frequency. When analyzing sound, we have to choose in what domain we should work, time or frequency but analysis can be combined. Fourier Transformations moving from time domain to frequency (and the opposite) are the standard mathematical tool for these operations. Of course when we are dealing with audio datasets we are dealing with digital representations of analogue audio signals which have digitized through sampling with specific sampling rate.



Sound can be represented as a waveform in a two dimensional representation of Amplitude and Time as Y and X axis.

Violin waveform

One of more interesting features that can be extracted from sound datasets is the spectrogram. The spectrogram is a frequnecy represntation in X axis and time in Y axis. A spectrogram is a visual representation of the spectrum of frequencies of sound or other signals as they vary with time. It's a representation of frequencies changing with respect to time for given music signals.



Violin spectrogram

1. Spectograms

Spectograms proved to be very important for the task of sound classification because the spectrogram images are the base for image classification that is used for transfer learning. Our approach of feature extraction of digital audio datasets is build upon the concepts of spectrograms, MEL spectrograms and their derivatives such as MFCC spectrograms and Chromagrams.

1.1 Spectograms

A spectrogram is an optical representation of a signal strength, or “loudness”, over time at various frequencies that are present in a particular waveform. In a spectrogram one can observe in what frequency is more or less energy at and can also see how energy levels vary over time. Spectograms are widely used to display frequencies of sound waves in audio analysis but has many other applications like seismic signal representation.

Spectrograms are basically two-dimensional graphs, with a third dimension represented by colors. Time runs from left to right along the horizontal axis. The vertical axis represents frequency, which can also be thought of as pitch or tone, with the lowest frequencies at the bottom and the highest frequencies at the top. The amplitude (“loudness”) of a particular frequency at a particular time is represented by the third dimension, color, with dark blues corresponding to low amplitudes and brighter colors up through red corresponding to progressively stronger (or louder) amplitudes.

1.2 MEL scale and MEL Spectograms

"The mel scale reflects how people hear musical tone"

The human auditory system doesn't interpret pitch in a linear manner. The human interpretation of the pitch changes with different frequencies, which in some applications may be a unwanted feature. To compensate for this the MEL scale was delevoped by Stevens, Volkmann, and Newman in 1937. The MEL scale was developed by experimenting with the human ears interpretation of a pitch in 1940's. The experiment showed that the pitch is lineary perceived in the frequency range 0-1000hz. Above 1000 hz, the scale becomes logarithmic. Umesh et al in 1999 were based on the aforementioned research and presented an updated formula of the MEL scale which is has been widely used since nowadays. [2]

Umesh et al. 1999 MEL scale data from Stevens and Volkmann 1940

Hz	40	161	200	404	693	867	1000	2022	3000	3393	4109	5526	6500	7743	12000
mel	43	257	300	514	771	928	1000	1542	2000	2142	2314	2600	2771	2914	3228

https://en.wikipedia.org/wiki/Mel_scale

Tip: The name **mel** comes from the word **melody** to indicate that the scale is based on pitch comparisons.

A special version of spectrograms called the MEL spectrograms are the ones that image classification techniques are applied on in order to achieve sound classification. MEL spectrograms are based on Mel scale as mentioned above. The MEL scale is applied as filter in real spectrograms thus producing MEL spectrograms.

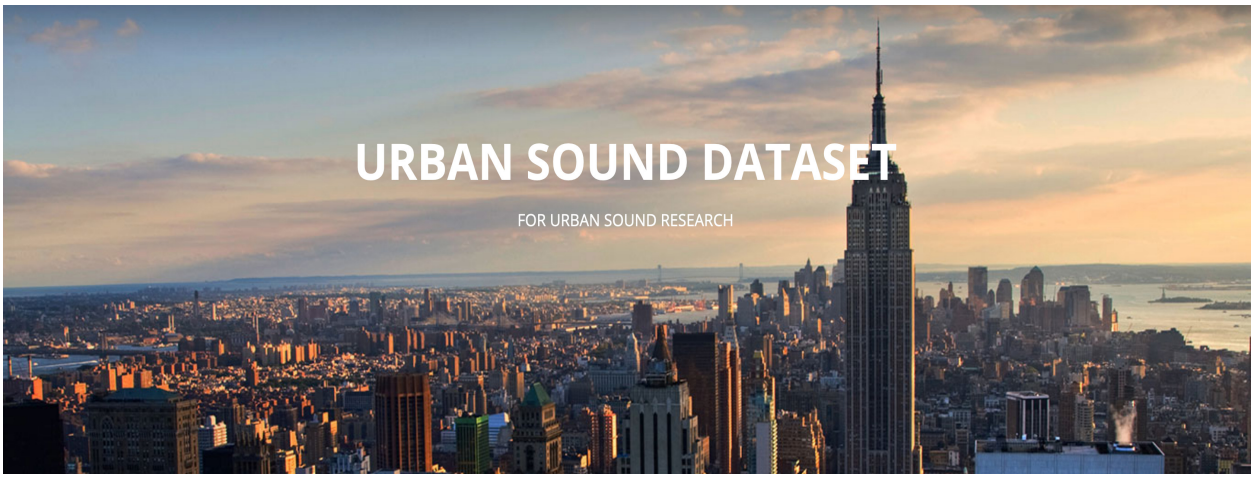
1.3 MFCC Spectograms

Mel scale and Mel spectrograms are also the base for Mel-frequency cepstral coefficients (MFCCs) which were considered to be the standard feature extraction from audio. MFCCs are coefficients are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale.

1.4 Chromagrams

Chroma features are an interesting and powerful representation for music audio in which the entire spectrum is projected onto bins representing the distinct semitones (or chromas) of the musical octave. The chroma representation tells us the intensity of each of the distinct musical chroma of the octave at each time frame.

2. Urban Sound Dataset and Classification Approach



2.1 The UrbanSound8k dataset

The urban sound dataset is build around the most frequent sound sources in noise complaints filed in New York City. There are 10 classes in the dataset:

1. air conditioner,
2. car horn
3. children playing
4. dog bark
5. drilling
6. engine idling
7. gun shot
8. jackhammer
9. siren
10. street music.

The source of the recordings is Freesound and it contains 8732 audio snippets of 4 seconds across each event class.

2.2 Classification approach

Our classification approach is to produce MEL/MFCC spectrograms and Chromagrams from the audio dataset and perform image classification techniques through convolutional neural networks. It may seem not intuitive to utilize convolutional neural networks to perform sound classification, but the theory is actually quite simple—all audio can be represented with spectrogram/chromagram images, depicting changes in frequency (Hz) and intensity/loudness

(dB) over time. If the group of sounds considered are individually distinct, their spectrogram plots will appear different enough to allow for distinguishing with a CNN.

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. If a group of sounds are individually distinct, their spectrogram plots will appear different enough to allow for distinguishing with a CNN. Since our sound clips are transformed into images we can proceed with image classification

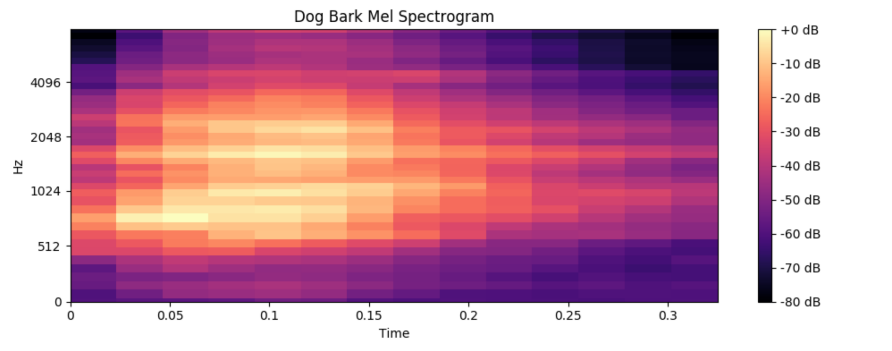
2.3 Feature extraction example

Below we have examples of the plots of our feature extractions of a specific audio clip from our dataset that lies in fold5.

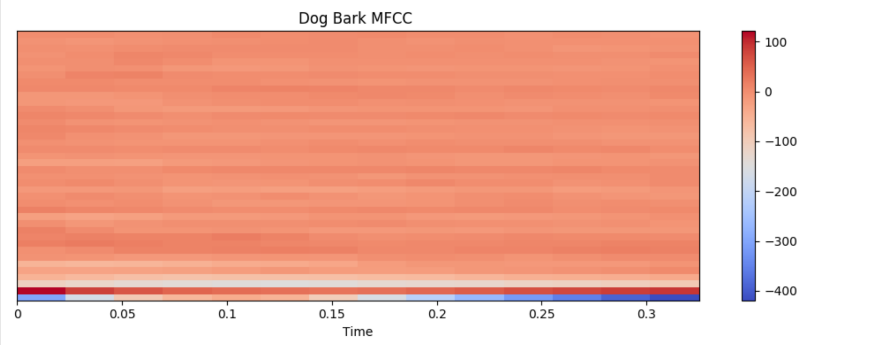
A DOG BARKS V1

100032-3-0-0.wav

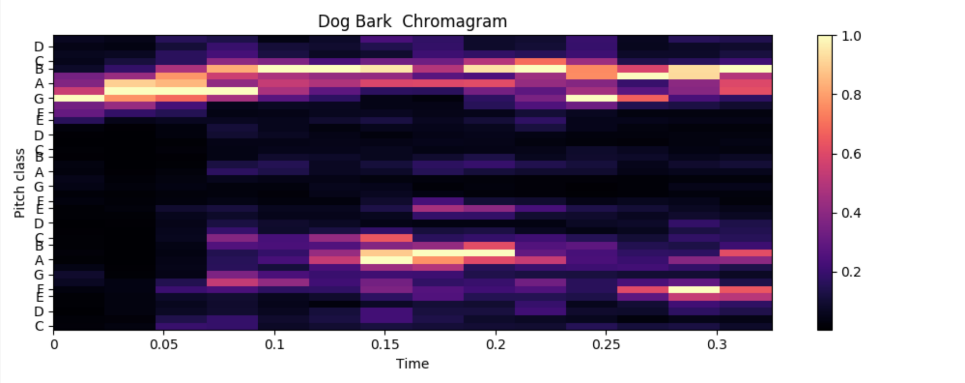
A DOG BARKS V2



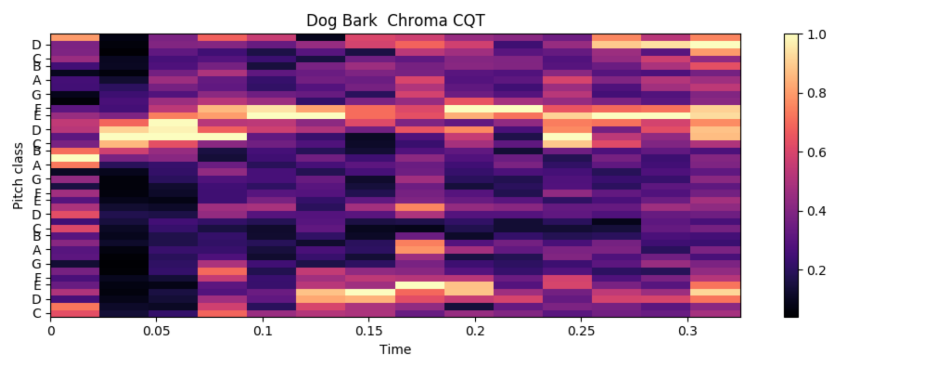
A DOG BARKS V3



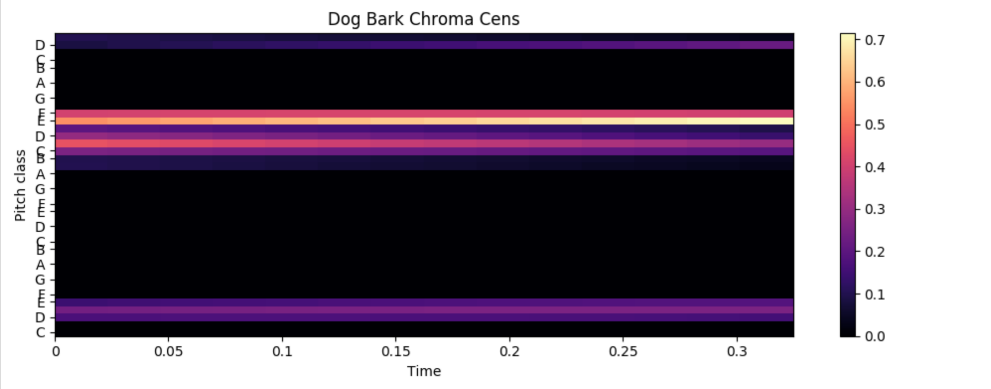
A DOG BARKS V4.1



A DOG BARKS V4.2



A DOG BARKS V4.3



3. CNN design and code implementation

3.1 Cnn design

As it is shown in the code we use the librosa python package for the feature extraction of audio clips. All features that are extracted produce matrices of shape 40x174. With this basic characteristic in mind we chose to apply Conv2d layers that start with small filter of 16 and grow as convolutional layers are added to 32, 64, 128. The kernel size is 2 as spectrogram images is suited for small masking window. The design was concluded after some first basic experimentation of 2 convolutional layers and no activation pools and dropouts to a four Conv2D layers with ReLu activation function as its linearity/cost is the most performant and widely used. We also added in each layer an activation pool of 2D for dimension reduction and better performance-fine tuning. After the last convolution layer a Dense fully connected layer was added as an output of the model that matches the number of the classes of the taxonomy of our dataset with a softmax activation function which is suitable for output layers. Lastly we used in our concluded CNN design a tuning of 150 epochs what improved our results without overfitting and Adam optimizer that is one of the most optimal optimizers for neural networks. This base architecture was used in slightly altered version for better matching each experimentation phase as it is discussed later on in more detail for each experiment.

Libraries and tools

The tools we used for our code implementation were librosa python package for sound analysis. The librosa library was the building block of our implementation as it provided the feature extraction of the sound clips of our dataset. The features that we extracted from the soundclips with librosa were:

- librosa.feature.melspectrogram - The Mel spectrogram of the audio clip
- librosa.feature.mfcc - The Mel Cepstrum Coefficients
- librosa.feature.chroma_stft - ChromaGram with short time fouriet transformation*
- librosa.feature.chroma_cqt - Chromagram with constant Q transformation
- librosa.feature.chroma_cens - Chromagaram with energy normalized statistics

For the CNN implementation we used the Keras high-level neural network API along with a tensorflow backend. Both tools are the industry/community leading frontend and backend respectively and served us build our CNN quite efficiently.

Finally we used scikit's sklearn python package for its great and community standard train/test splitting of our dataset.

*librosa stft feature extraction produced many warnings in the specific dataset so we excluded this feature from our experiments.

3.2 Experimentation phases

Our first experimentation phase was to use our first simple CNN design with 2 layers, without fine tuning and 50 epochs for each feature extraction separately. We applied this model in all feature extractions and it turned out that MFCC spectrograms provided the best results with 73% accuracy. This was quite expected as the MFCC feature extraction is considered to be the most powerful feature extraction for sound analysis because of its cepstral representation of the equally divided mel scaled frequencies.

Based on this experiment and with MFCC spectrograms as a base case we proceeded our fine tuning and redesign of our cnn architecture as described above in order to achieve the best accuracy. After concluding to the CNN architecture and parameter fine tuning we achieved an accuracy rate of 99% as we will discuss below in detail. Having our final design we ended up with three main experimentation phases. The first was to feed the neural network with each feature extraction separately in order to observe which one could give a good performance in terms of accuracy. The second was an attempt of concatenation of all feature extractions together whereas the third was to concatenate different CNN models inputs of all feature extractions separately and then merge them in one model and common output.

3.2.1 Separate feature extraction experiments

As described in section 3.2 we fed the neural network with MFCC spectrograms, MEL spectrograms and Cqt. As it is shown in the code the parameters were chosen in a way that the returned matrices would be of the same shape. The most basic parameter for all feature extractions (n_mels, n_mfccs, n_chromas) was the number 40 because our intention was to make improvements upon the original classification paper [1] that had the same parameter tuning. Thus all our matrices had a number of rows according to the above tuning (40). It was observed that the returned matrices were of different size as the clips are all under 4 second but are not all of exact same length. Specifically the longest matrix was of [40, 173] shape. To tackle this we added the appropriate padding in all the returned matrices in order to have all the matrices of the same shape.

We performed measurements as our code instructed in each feature extraction separately and the results were:

	Training Accuracy	**Testing Accuracy**
MFCC	0.9896921975662133	0.9307384085300382
MEL	0.9342877594931431	0.8729250145490741
CQT	0.9899785254115963	0.9084144247281053

As shown in the original UrbanSound Dataset paper[1] we observe the results

of image classification that was performed in MFCC spectrograms with parameter n_mfcc to be equal to 40. The results show that a baseline average accuracy of 68% was achieved. We should note here that the audio clips were were larger than 4s. As a latter approach the authors of the original urban sound paper[1] trimmed the original sound clips to the form of UrbanSound8k dataset with <4s sound clips.

3.2.2 Concatenation of feature extractions

Another experimentation we conducted was to concatenate the returned matrices of the feature extractions on one large matrix of the MFCC, MEL, CQT and CENS features. Our results from this experiment approach was our second best after separate MFCC:

	Training Accuracy	**Testing Accuracy**
MFCC/MEL/CQT/CENS	0.9755189692197567	0.92043503131195

[*] It has already been pointed to us during our presentation that with this experiment an amount of noise is inserted to our feature data as the concatenation of the image matrices will be overlapped at the joint point of any two images with the convolution window at the convolution procedure.

3.2.3 Concatenation of separate CNN models

Our final experimentation dealt with another approach, that of concatenating the separate models of each feature extraction of MFCC, MEL, Chroma CQT and Chroma CENS. We extracted the matrices of each feature separately, added a dense layer in each model and then we concatenated the inputs. The concatenated input were converted to a single input of a final merged model and we produced a dense final output as in previous CNN architectures of 10 layers that are the number of our label classes. Keras framework provided us very useful concatenate and merge functionality that helped us achieve this multiplexing of separate models. The results of this experiment were:

	Training Accuracy	**Testing Accuracy**
MFCC/MEL/CQT/CENS	0.9941302791696492	0.9318832286303563

As we see above in the results with this approach we achieved the best accuracy result.

4. Code

We provide below the python code of our implementation. We divided our code in different python programs for each main experiment for understandability purposes. The different python codes are in different files to avoid boilerplate code. Lastly in each code file there is a part of predicting an unlabeled audio dataset from the original UrbanSound dataset which was a smaller one. This part was for fun purposes and it is given as it run at our machine!

4.1 MFCC/MEL/Chroma separate feature extraction

We present here only the MFCC separate feature extraction code because all the other versions of code have one line of code different which is what feature is extracted. The rest feature extractions are on comment.

```
import librosa
import pandas as pd
from tqdm import tqdm
import numpy as np
import numpy as np
import matplotlib.pyplot as plt
from librosa import display
from pathlib import Path

# Extract selected audio features from an audio file
def extract_audio_feature(audio_file):
    pad = 174
    y, sr = librosa.load(audio_file, res_type='kaiser_fast')
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)
    # mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=40, fmax=8000)
    # chroma_cq = librosa.feature.chroma_cqt(y=y, sr=sr, n_chroma=40)
    new_pad = pad - mfccs.shape[1]
    return np.pad(mfccs, pad_width=((0, 0), (0, new_pad)), mode='constant')

# Predict and print class of an unlabeled audio file
def predict_class_of_audio_file(audio_file):
    pred = extract_audio_feature(audio_file)
    pred = pred.reshape(1, num_rows, num_columns, num_channels)
    pred_class = model.predict_classes(pred)
    pred_class_trans = le.inverse_transform(pred_class)
    print("The predicted class is:", pred_class_trans[0])

# Read metadata of urban sound dataset as pandas dataframe
metadata = pd.read_csv('/home/dimitris/Desktop/UrbanSound8K/metadata/UrbanSound8K.csv')

print(metadata.head())
print(metadata.class_name.value_counts())

# Fill up feature_list List with features extraction of all audio files
path="/home/dimitris/Desktop/UrbanSound8K/audio/fold"
feature_list = []
for i in tqdm(range(len(metadata))):
    fold_no = str(metadata.iloc[i]["fold"])
    file = metadata.iloc[i]["slice_file_name"]
    class_label_id = metadata.iloc[i]["classID"]
    class_label = metadata.iloc[i]["class_name"]
    filename = path+fold_no+"/"+file
    data = extract_audio_feature(filename)
    feature_list.append([data, class_label])
```

```
# Convert feature_list to pandas dataframe
features_dataframe = pd.DataFrame(feature_list, columns=['feature', 'class_label'])

from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical

# Convert features and class labels to numpy arrays
X = np.array(features_dataframe.feature.tolist())
y = np.array(features_dataframe.class_label.tolist())

# Encode classification labels
le = LabelEncoder()
categorical_y = to_categorical(le.fit_transform(y))

from sklearn.model_selection import train_test_split

# Split features array to train and test sets
x_train, x_test, y_train, y_test = train_test_split(X, categorical_y, test_size=0.2, random_state
= 42)

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
#
# from keras import backend as K
# K.tensorflow_backend._get_available_gpus()

num_rows = 40
num_columns = 174
num_channels = 1

x_train = x_train.reshape(x_train.shape[0], num_rows, num_columns, num_channels)
x_test = x_test.reshape(x_test.shape[0], num_rows, num_columns, num_channels)

num_labels = categorical_y.shape[1]
filter_size = 2

# CNN model architecture
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=2, input_shape=(num_rows, num_columns, num_channels),
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=32, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=64, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
```

```
model.add(Conv2D(filters=128, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(GlobalAveragePooling2D())

model.add(Dense(num_labels, activation='softmax'))

# CNN model compilation
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')

# CNN model summary
model.summary()

# Train CNN model
model.fit(x_train, y_train, batch_size=256, epochs=150, validation_data=(x_test, y_test),
verbose=1)

# Evaluate CNN model
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])

model.save('/home/dimitris/Desktop/UrbanSound8K/urban_sound_cnn_model1.h5')

# Predict unlabeled audio files
from pathlib import Path
import os
path_str="/home/dimitris/Desktop/urban_sound_dataset/test2"
pathlist = Path(path_str).glob('**/*.wav')
for path in pathlist:
     audio_file_path = str(path)
     head, tail = os.path.split(audio_file_path)
     print(tail)
     predict_class_of_audio_file(audio_file_path)
     print("\n")
```

4.2 Concatenation of MFCC/MEL/Chroma feature extraction

We present here the MFCC/MEL/Chroma concatenated feature extractions code. Although the biggest part of the codebase is similar we decided to have a different python file to be more understandable for the whole concatenation and different padding and shaping of the input to the CNN.

```
import librosa
import pandas as pd
from tqdm import tqdm
import numpy as np
import numpy as np
```

```
import matplotlib.pyplot as plt
from librosa import display
from pathlib import Path

# Extract selected audio features from an audio file
def extract_audio_feature(audio_file):
    pad = 174
    y, sr = librosa.load(audio_file, res_type='kaiser_fast')
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)
    new_pad = pad - mfccs.shape[1]
    mfccs =  np.pad(mfccs, pad_width=((0, 0), (0, new_pad)), mode='constant')
    mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=40, fmax=8000)
    new_pad = pad - mel_spectrogram.shape[1]
    mel_spectrogram = np.pad(mel_spectrogram, pad_width=((0, 0), (0, new_pad)), mode='constant')
    chroma_cq = librosa.feature.chroma_cqt(y=y, sr=sr, n_chroma=40)
    new_pad = pad - chroma_cq.shape[1]
    chroma_cq = np.pad(chroma_cq, pad_width=((0, 0), (0, new_pad)), mode='constant')
    chroma_cens = librosa.feature.chroma_cens(y=y, sr=sr, n_chroma=40)
    new_pad = pad - chroma_cens.shape[1]
    chroma_cens = np.pad(chroma_cens, pad_width=((0, 0), (0, new_pad)), mode='constant')
    return np.concatenate((mfccs, mel_spectrogram, chroma_cq, chroma_cens))

# Predict and print class of an unlabeled audio file
def predict_class_of_audio_file(audio_file):
    pred = extract_audio_feature(audio_file)
    pred = pred.reshape(1, num_rows, num_columns, num_channels)
    pred_class = model.predict_classes(pred)
    pred_class_trans = le.inverse_transform(pred_class)
    print("The predicted class is:", pred_class_trans[0])

# Read metadata of urban sound dataset as pandas dataframe
metadata = pd.read_csv('/home/dimitris/Desktop/UrbanSound8K/metadata/UrbanSound8K.csv')

print(metadata.head())
print(metadata.class_name.value_counts())

# Fill up feature_list List with features extraction of all audio files
path="/home/dimitris/Desktop/UrbanSound8K/audio/fold"
feature_list = []
for i in tqdm(range(len(metadata))):
    fold_no = str(metadata.iloc[i]["fold"])
    file = metadata.iloc[i]["slice_file_name"]
    class_label_id = metadata.iloc[i]["classID"]
    class_label = metadata.iloc[i]["class_name"]
    filename = path+fold_no+"/"+file
    data = extract_audio_feature(filename)
    feature_list.append([data, class_label])


# Convert feature_list to pandas dataframe
features_dataframe = pd.DataFrame(feature_list, columns=['feature', 'class_label'])

from sklearn.preprocessing import LabelEncoder
```

```
from keras.utils import to_categorical

# Convert features and class labels to numpy arrays
X = np.array(features_dataframe.feature.tolist())
y = np.array(features_dataframe.class_label.tolist())

# Encode classification labels
le = LabelEncoder()
categorical_y = to_categorical(le.fit_transform(y))

from sklearn.model_selection import train_test_split

# Split features array to train and test sets
x_train, x_test, y_train, y_test = train_test_split(X, categorical_y, test_size=0.2, random_state
= 42)

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
num_rows = 160
num_columns = 174
num_channels = 1

x_train = x_train.reshape(x_train.shape[0], num_rows, num_columns, num_channels)
x_test = x_test.reshape(x_test.shape[0], num_rows, num_columns, num_channels)

num_labels = categorical_y.shape[1]
filter_size = 2

# CNN model architecture
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=2, input_shape=(num_rows, num_columns, num_channels),
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=32, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=64, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(GlobalAveragePooling2D())

model.add(Dense(num_labels, activation='softmax'))

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

```
# CNN model summary
model.summary()

# Train CNN model
model.fit(x_train, y_train, batch_size=256, epochs=150, validation_data=(x_test, y_test),
verbose=1)

# Evaluate CNN model
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])

model.save('/home/dimitris/Desktop/UrbanSound8K/urban_sound_cnn_model2.h5')

# Predict unlabeled audio files
from pathlib import Path
import os

path_str="/home/dimitris/Desktop/urban_sound_dataset/test2"
pathlist = Path(path_str).glob('**/*.wav')
for path in pathlist:
     audio_file_path = str(path)
     head, tail = os.path.split(audio_file_path)
     print(tail)
     predict_class_of_audio_file(audio_file_path)
     print("\n")
```

4.3 MFCC/MEL/Chroma concatenated models code

We present here the MFCC/MEL/Chroma concatenated models code. This code presents an alternate approach of a CNN design and implementation. In this code we extracted each feature separately and gave it as input to a separate CNN model with the same characteristics as before. The only difference is that we added at the end of each separate model a dense layer in order for the merging to be more subtle as it was widely suggested in the literature and programming communities.

```
import librosa
import pandas as pd
from tqdm import tqdm
import numpy as np
import numpy as np
import matplotlib.pyplot as plt
from librosa import display
from pathlib import Path

# Extract mfcc audio features from an audio file
def extract_mfcc_audio_feature(audio_file):
    pad = 174
```

```
    y, sr = librosa.load(audio_file, res_type='kaiser_fast')
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)
    new_pad = pad - mfccs.shape[1]
    mfccs =  np.pad(mfccs, pad_width=((0, 0), (0, new_pad)), mode='constant')
    return mfccs

# Extract mel audio features from an audio file
def extract_mel_audio_feature(audio_file):
    pad = 174
    y, sr = librosa.load(audio_file, res_type='kaiser_fast')
    mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=40, fmax=8000)
    new_pad = pad - mel_spectrogram.shape[1]
    mel_spectrogram = np.pad(mel_spectrogram, pad_width=((0, 0), (0, new_pad)), mode='constant')
    return mel_spectrogram

# Extract chroma cqt audio features from an audio file
def extract_cqt_audio_feature(audio_file):
    pad = 174
    y, sr = librosa.load(audio_file, res_type='kaiser_fast')
    chroma_cq =librosa.feature.chroma_cqt(y=y, sr=sr, n_chroma=40)
    new_pad = pad - chroma_cq.shape[1]
    chroma_cq = np.pad(chroma_cq, pad_width=((0, 0), (0, new_pad)), mode='constant')
    return chroma_cq

# Extract chroma cens audio features from an audio file
def extract_cens_audio_feature(audio_file):
    pad = 174
    y, sr = librosa.load(audio_file, res_type='kaiser_fast')
    chroma_cens =librosa.feature.chroma_cens(y=y, sr=sr, n_chroma=40)
    new_pad = pad - chroma_cens.shape[1]
    chroma_cens = np.pad(chroma_cens, pad_width=((0, 0), (0, new_pad)), mode='constant')
    return chroma_cens


# Read metadata of urban sound dataset as pandas dataframe
metadata = pd.read_csv('/home/dimitris/Desktop/UrbanSound8K/metadata/UrbanSound8K'
                       '.csv')

print(metadata.head())
print(metadata.class_name.value_counts())

# Fill up feature_list List with features extraction of all audio files
path="/home/dimitris/Desktop/UrbanSound8K/audio/fold"

mfcc_feature_list = []
mel_feature_list = []
cqt_feature_list = []
cens_feature_list = []

for i in tqdm(range(len(metadata))):
    fold_no = str(metadata.iloc[i]["fold"])
    file = metadata.iloc[i]["slice_file_name"]
    class_label_id = metadata.iloc[i]["classID"]
```

```
    class_label = metadata.iloc[i]["class_name"]
    filename = path+fold_no+"/"+file
    mfcc_data = extract_mfcc_audio_feature(filename)
    mel_data = extract_mel_audio_feature(filename)
    cqt_data = extract_cqt_audio_feature(filename)
    cens_data = extract_cens_audio_feature(filename)
    mfcc_feature_list.append([mfcc_data, class_label])
    mel_feature_list.append([mel_data, class_label])
    cqt_feature_list.append([cqt_data, class_label])
    cens_feature_list.append([cens_data, class_label])


# Convert mfcc feature_list to pandas dataframe
features_dataframe = pd.DataFrame(mfcc_feature_list, columns=['feature', 'class_label'])

from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical

# Convert features and class labels to numpy arrays
X = np.array(features_dataframe.feature.tolist())
y = np.array(features_dataframe.class_label.tolist())

# Encode classification labels
le = LabelEncoder()
categorical_y = to_categorical(le.fit_transform(y))

from sklearn.model_selection import train_test_split

# Split features array to train and test sets
x_train_mfcc, x_test_mfcc, y_train_mfcc, y_test_mfcc = train_test_split(X, categorical_y,
test_size=0.2, random_state = 42)

# Convert mel feature_list to pandas dataframe
features_dataframe = pd.DataFrame(mel_feature_list, columns=['feature', 'class_label'])


# Convert features and class labels to numpy arrays
X = np.array(features_dataframe.feature.tolist())
y = np.array(features_dataframe.class_label.tolist())

# Encode classification labels
le = LabelEncoder()
categorical_y = to_categorical(le.fit_transform(y))

# Split features array to train and test sets
x_train_mel, x_test_mel, y_train_mel, y_test_mel = train_test_split(X, categorical_y,
test_size=0.2, random_state = 42)

# Convert cqt feature_list to pandas dataframe
features_dataframe = pd.DataFrame(cqt_feature_list, columns=['feature', 'class_label'])

# Convert features and class labels to numpy arrays
X = np.array(features_dataframe.feature.tolist())
```

```
y = np.array(features_dataframe.class_label.tolist())

# Encode classification labels
le = LabelEncoder()
categorical_y = to_categorical(le.fit_transform(y))

# Split features array to train and test sets
x_train_cqt, x_test_cqt, y_train_cqt, y_test_cqt = train_test_split(X, categorical_y,
test_size=0.2, random_state = 42)

# Convert cens feature_list to pandas dataframe
features_dataframe = pd.DataFrame(cqt_feature_list, columns=['feature', 'class_label'])

# Convert features and class labels to numpy arrays
X = np.array(features_dataframe.feature.tolist())
y = np.array(features_dataframe.class_label.tolist())

# Encode classification labels
le = LabelEncoder()
categorical_y = to_categorical(le.fit_transform(y))

# Split features array to train and test sets
x_train_cens, x_test_cens, y_train_cens, y_test_cens = train_test_split(X, categorical_y,
test_size=0.2, random_state = 42)

from keras.models import Sequential, Model
from keras.layers.merge import concatenate
from keras.layers import Dense, Dropout, merge
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D

num_rows = 40
num_columns = 174
num_channels = 1

x_train_mfcc = x_train_mfcc.reshape(x_train_mfcc.shape[0], num_rows, num_columns, num_channels)
x_test_mfcc = x_test_mfcc.reshape(x_test_mfcc.shape[0], num_rows, num_columns, num_channels)

x_train_mel = x_train_mel.reshape(x_train_mel.shape[0], num_rows, num_columns, num_channels)
x_test_mel = x_test_mel.reshape(x_test_mel.shape[0], num_rows, num_columns, num_channels)

x_train_cqt = x_train_cqt.reshape(x_train_cqt.shape[0], num_rows, num_columns, num_channels)
x_test_cqt = x_test_cqt.reshape(x_test_cqt.shape[0], num_rows, num_columns, num_channels)

x_train_cens = x_train_cens.reshape(x_train_cens.shape[0], num_rows, num_columns, num_channels)
x_test_cens = x_test_cens.reshape(x_test_cens.shape[0], num_rows, num_columns, num_channels)

num_labels = categorical_y.shape[1]

# Mfcc CNN model architecture
mfcc_model = Sequential()
mfcc_model.add(Conv2D(filters=16, kernel_size=2, input_shape=(num_rows, num_columns,
num_channels), activation='relu'))
mfcc_model.add(MaxPooling2D(pool_size=2))
```

```
mfcc_model.add(Dropout(0.2))

mfcc_model.add(Conv2D(filters=32, kernel_size=2, activation='relu'))
mfcc_model.add(MaxPooling2D(pool_size=2))
mfcc_model.add(Dropout(0.2))

mfcc_model.add(Conv2D(filters=64, kernel_size=2, activation='relu'))
mfcc_model.add(MaxPooling2D(pool_size=2))
mfcc_model.add(Dropout(0.2))

mfcc_model.add(Conv2D(filters=128, kernel_size=2, activation='relu'))
mfcc_model.add(MaxPooling2D(pool_size=2))
mfcc_model.add(Dropout(0.2))
mfcc_model.add(GlobalAveragePooling2D())

mfcc_model.add(Dense(256, activation='relu'))

# Mel CNN model architecture
mel_model = Sequential()
mel_model.add(Conv2D(filters=16, kernel_size=2, input_shape=(num_rows, num_columns,
num_channels), activation='relu'))
mel_model.add(MaxPooling2D(pool_size=2))
mel_model.add(Dropout(0.2))

mel_model.add(Conv2D(filters=32, kernel_size=2, activation='relu'))
mel_model.add(MaxPooling2D(pool_size=2))
mel_model.add(Dropout(0.2))

mel_model.add(Conv2D(filters=64, kernel_size=2, activation='relu'))
mel_model.add(MaxPooling2D(pool_size=2))
mel_model.add(Dropout(0.2))

mel_model.add(Conv2D(filters=128, kernel_size=2, activation='relu'))
mel_model.add(MaxPooling2D(pool_size=2))
mel_model.add(Dropout(0.2))
mel_model.add(GlobalAveragePooling2D())

mel_model.add(Dense(256, activation='relu'))

# Cqt CNN model architecture
cqt_model = Sequential()
cqt_model.add(Conv2D(filters=16, kernel_size=2, input_shape=(num_rows, num_columns,
num_channels), activation='relu'))
cqt_model.add(MaxPooling2D(pool_size=2))
cqt_model.add(Dropout(0.2))

cqt_model.add(Conv2D(filters=32, kernel_size=2, activation='relu'))
cqt_model.add(MaxPooling2D(pool_size=2))
cqt_model.add(Dropout(0.2))

cqt_model.add(Conv2D(filters=64, kernel_size=2, activation='relu'))
cqt_model.add(MaxPooling2D(pool_size=2))
cqt_model.add(Dropout(0.2))
```

```
cqt_model.add(Conv2D(filters=128, kernel_size=2, activation='relu'))
cqt_model.add(MaxPooling2D(pool_size=2))
cqt_model.add(Dropout(0.2))
cqt_model.add(GlobalAveragePooling2D())

cqt_model.add(Dense(256, activation='relu'))

# Cens CNN model architecture
cens_model = Sequential()
cens_model.add(Conv2D(filters=16, kernel_size=2, input_shape=(num_rows, num_columns,
num_channels), activation='relu'))
cens_model.add(MaxPooling2D(pool_size=2))
cens_model.add(Dropout(0.2))

cens_model.add(Conv2D(filters=32, kernel_size=2, activation='relu'))
cens_model.add(MaxPooling2D(pool_size=2))
cens_model.add(Dropout(0.2))

cens_model.add(Conv2D(filters=64, kernel_size=2, activation='relu'))
cens_model.add(MaxPooling2D(pool_size=2))
cens_model.add(Dropout(0.2))

cens_model.add(Conv2D(filters=128, kernel_size=2, activation='relu'))
cens_model.add(MaxPooling2D(pool_size=2))
cens_model.add(Dropout(0.2))
cens_model.add(GlobalAveragePooling2D())

cens_model.add(Dense(256, activation='relu'))

# Concatenation of models
concat = concatenate([mfcc_model.output, mel_model.output, cqt_model.output, cens_model.output])
concat_out = Dense(num_labels, activation='softmax')(concat)

# Merge models
merge_model = Model([mfcc_model.input, mel_model.input, cqt_model.input, cens_model.input],
concat_out)

# CNN model compilation
merge_model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')

# CNN model summary
merge_model.summary()

# Train CNN model
merge_model.fit([x_train_mfcc, x_train_mel, x_train_cqt, x_train_cens], y_train_mfcc,
batch_size=256, epochs=150, validation_data=([x_test_mfcc, x_test_mel, x_test_cqt, x_test_cens],
y_test_mfcc), verbose=1)

# Evaluate CNN model
score = merge_model.evaluate([x_train_mfcc, x_train_mel, x_train_cqt, x_train_cens],
y_train_mfcc, verbose=0)
print("Training Accuracy: ", score[1])
```

```
score = merge_model.evaluate([x_test_mfcc, x_test_mel, x_test_cqt, x_test_cens], y_test_mfcc,
verbose=0)
print("Testing Accuracy: ", score[1])

merge_model.save('/home/dimitris/Desktop/UrbanSound8K/urban_sound_cnn_model3.h5')
```

4.4 Code for plotting

Below is the code for plotting the spectrograms and chromagrams that were presented in section 2.3.

```
from librosa import display
import librosa
import numpy as np
import matplotlib.pyplot as plt
import os


# Feature extraction of dog bark
y,sr=librosa.load("/Users/dimitris/Desktop/urban_sound_dataset/UrbanSound8K/audio/fold5/100032-3-
0-0.wav")
mfccs = librosa.feature.mfcc(y, sr, n_mfcc=40)
mel_spectrogram =librosa.feature.melspectrogram(y=y, sr=sr, n_mels=40, fmax=8000)
chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr, n_chroma=40)
chroma_cq = librosa.feature.chroma_cqt(y=y, sr=sr, n_chroma=40)
chroma_cens = librosa.feature.chroma_cens(y=y, sr=sr, n_chroma=40)
print(mel_spectrogram.shape, chroma_stft.shape, chroma_cq.shape, chroma_cens.shape, mfccs.shape)

#MFCC of dog bark
import matplotlib.pyplot as plt
plt.figure(figsize=(10,4))
librosa.display.specshow(mfccs, x_axis='time')
plt.colorbar()
plt.title('Dog Bark MFCC ')
plt.tight_layout()
plt.show()

#Melspectrogram of a dog bark
plt.figure(figsize=(10,4))
librosa.display.specshow(librosa.power_to_db(mel_spectrogram, ref=np.max), y_axis='mel',
fmax=8000, x_axis='time')
plt.colorbar(format='%+2.0f dB')
plt.title('Dog Bark Mel Spectrogram')
plt.tight_layout()
plt.show()

#Chromagram of dog bark
plt.figure(figsize=(10,4))
librosa.display.specshow(chroma_stft, y_axis='chroma', x_axis='time')
plt.colorbar()
plt.title('Dog Bark  Chromagram')
```

```
plt.tight_layout()
plt.show()

#Chroma cqt of a dog bark
plt.figure(figsize=(10,4))
librosa.display.specshow(chroma_cq, y_axis='chroma', x_axis='time')
plt.colorbar()
plt.title('Dog Bark  Chroma CQT')
plt.tight_layout()
plt.show()

#Chroma cens of a dog bark
plt.figure(figsize=(10,4))
librosa.display.specshow(chroma_cens, y_axis='chroma', x_axis='time')
plt.colorbar()
plt.title('Dog Bark Chroma Cens')
plt.tight_layout()
plt.show()
```


[1]http://www.justinsalamon.com/uploads/4/3/9/4/4394963/salamon_urbansound_acmmm14.pdf
[2]http://kom.aau.dk/group/04gr742/pdf/MFCC_worksheet.pdf
[3]http://karol.piczak.com/papers/Piczak2015-ESC-ConvNet.pdf
[4]https://arxiv.org/pdf/1609.09430.pdf
[5]https://arxiv.org/pdf/1802.02617.pdf