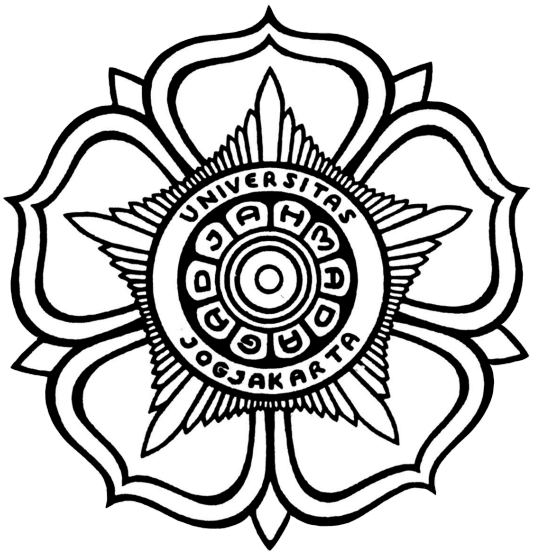
Application of MFCC Feature Extraction for Urban City Noises Classification



Sasha Annabel
22/496780/PA/21361

PATTERN RECOGNITION SUBJECT
FACULTY OF MATHEMATICS AND NATURAL SCIENCES
UNIVERSITAS GADJAH MADA

LIST OF CONTENTS

I. INTRODUCTION

The busy urban environment is nearly always filled with sounds and noises from its people's daily lives, whether from the clamor of traffic, yells of children playing in the park, sounds of construction work like drilling, or even emergency sirens. Recently, the problem of noise pollution is becoming more adamant as there is a growing number of modern metropolitan cities being built. Noise pollution poses significant disturbance to urban residents' quality of life and health, which is why it is decided that addressing such an issue will be a necessity.

This project aims to classify the various urban noises into 10 distinct categories; dog barking, drilling, children playing, air conditioner, car horn, engine idling, gun shot, jackhammer, siren, and street music. By categorizing urban sounds, the project seeks to provide valuable insights for urban planners, policymakers, and public health officials, hopefully enabling them to make informed decisions to mitigate noise pollution. In particular, for the purpose of city infrastructure planning. Ultimately, this classification framework aims to contribute to the creation of quieter and more livable urban environments, benefiting both current and future city dwellers.

II. METHODS

A. Libraries and Environment

The chosen environment of GoogleCOLAB and Python libraries play integral roles in facilitating various stages of the classification process:

- Librosa is the main library used for this audio-processing project. Apart from enabling efficient audio preprocessing and feature extraction, it is crucial for transforming raw audio data into a format suitable for analysis because librosa automatically **normalizes** the raw audio data into a standardized sample rate.
- Matplotlib provides robust visualization tools for exploring the characteristics of urban sounds, useful in the early stage of data exploration and interpretation.
- Numpy and pandas offer essential data manipulation capabilities, allowing for seamless handling and manipulation of large datasets.
- TensorFlow serves as the backbone for machine learning models, enabling the training and evaluation of classifiers to classify urban noises effectively.
- Lastly, scikit-learn offers a wide range of classification algorithms and evaluation metrics, useful for model selection and performance assessment.

Together, these tools contribute to the project's success in accurately classifying urban sounds into distinct categories.

B. Mel-Frequency Cepstral Coefficient (MFCC)

The Mel-Frequency Cepstral Coefficients (MFCC) method is employed to streamline the analysis of audio sounds. MFCC itself simplifies the complexity of audio data by breaking it down into key features and operates by segmenting the audio signal into small time frames. It converts each frame into a frequency-domain representation using Fast Fourier Transform (FFT), and then applies a filterbank that mimics human auditory sensitivity to different frequencies:

- MFCC will map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows or alternatively, cosine overlapping windows.
- It takes the logs of the powers at each of the mel frequencies.
- Then takes the discrete cosine transform of the list of mel log powers, as if it were a signal.
- The MFCCs are the amplitudes of the resulting spectrum.

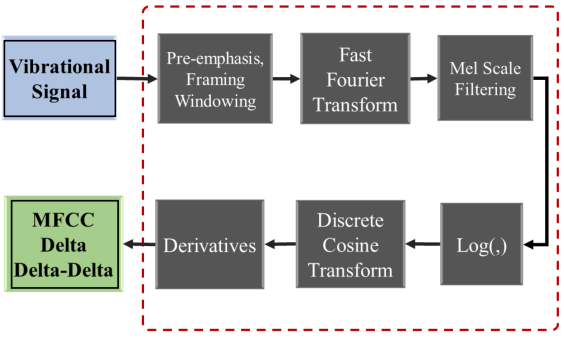


Fig 1. Example diagram of how the MFCC method works.

By extracting cepstral coefficients, which capture essential spectral characteristics, MFCC distills the audio's key attributes. This method has been found through much research that it is ideal for audio classification because it focuses on fundamental sound characteristics such as pitch and timbre, making it robust for distinguishing between various types of urban noises as needed in this project.

C. Artificial Neural Networks

Artificial Neural Networks (ANN) is a powerful tool to train the model that will be later used to classify the urban sounds. ANN adjusts the weights of connections between neurons from interconnected layers to minimize the difference between predicted and actual outputs during "*training",* which is the iterative learning process that enables ANN to identify patterns and relationships within the data. With ANN's ability to learn from data and generalize patterns, the model is hoped to be proficient in accurately classifying urban noises into the mentioned 10 distinct categories.

III. IMPLEMENTATIONS

A. Exploratory Data Analysis (EDA)

The Urban Sound audio database, taken from researchers at New York University, includes 8732 sound excerpts in *.wav* format. As shown in Fig. 2, each audio file has been labeled into one of the 10 category classes mentioned, and it can also be seen in Fig. 3 that the data is quite balanced for each of those classes, thus there is no need for alterations in that aspect.

```
[ ] import pandas as pd

    metadata = pd.read_csv('metadata/UrbanSound8K.csv')
    metadata.head(15) #read the first 15 data
```

	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.000000	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.500000	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.500000	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.000000	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.500000	72.500000	1	5	2	children_playing
5	100263-2-0-143.wav	100263	71.500000	75.500000	1	5	2	children_playing
6	100263-2-0-161.wav	100263	80.500000	84.500000	1	5	2	children_playing
7	100263-2-0-3.wav	100263	1.500000	5.500000	1	5	2	children_playing
8	100263-2-0-36.wav	100263	18.000000	22.000000	1	5	2	children_playing
9	100648-1-0-0.wav	100648	4.823402	5.471927	2	10	1	car_horn
10	100648-1-1-0.wav	100648	8.998279	10.052132	2	10	1	car_horn
11	100648-1-2-0.wav	100648	16.699509	17.104837	2	10	1	car_horn
12	100648-1-3-0.wav	100648	17.631764	19.253075	2	10	1	car_horn
13	100648-1-4-0.wav	100648	25.332994	27.197502	2	10	1	car_horn
14	100652-3-0-0.wav	100652	0.000000	4.000000	1	2	3	dog_bark

Fig 2. Preview of first 15 rows in Urban Sound dataset

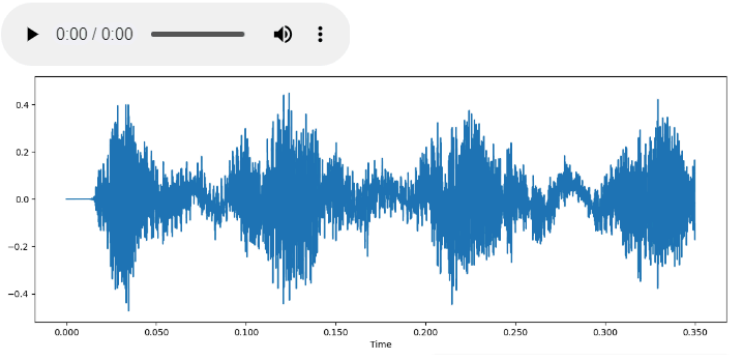
```
[ ] metadata['class'].value_counts()

    dog_bark            1000
    children_playing    1000
    air_conditioner     1000
    street_music        1000
    engine_idling       1000
    jackhammer          1000
    drilling            1000
    siren                929
    car_horn             429
    gun_shot             374
    Name: class, dtype: int64
```

Fig 3. Preview on distribution of amount of labeled data for each class

There were two main problems upon inspection, however. One, the sample rates for the audio data are not uniform, most audio has more than one channel (one dimension). Therefore, librosa library is used as it can automatically convert any audio data into being **monochannel (one-dimensional)** and having **uniform sample rate of 22050 kHz**.

```
[ ]  filename = '4201-3-2-0.wav'
```

```
[ ]  plt.figure(figsize=(14,5))
     data,sample_rate = librosa.load(filename)
     librosa.display.waveshow(data,sr=sample_rate)
     ipd.Audio(filename)
```




```
sample_rate
```

```
22050
```

Fig 4. Example output of audio file '4201-3-2-0.wav' after getting processed by librosa into being mono channel and having 22050 kHz sample rate, despite its raw data being 2 channels and having 11025 kHz sample rate

B. Audio Data Pre-Processing

As explained in Section II. Methods, MFCC is the method used to extract the features from each audio file, as it provides a compact representation of the spectral envelope of the signal while discarding irrelevant details (background noise, for instance). For this case, the features extracted have been decided to be as many as 40, to ensure that the uniqueness of audios from each class can be captured well.

```
[18] def extract_features(file):
       audio_results, sample_rate_results = librosa.load(audio_file_name,
       res_type='kaiser_fast')

       mfccs_features = librosa.feature.mfcc(y = audio_results,
       sr = sample_rate_results, n_mfcc = 40)
       mfccs_features_final = np.mean(mfccs_features.T, axis=0)
       return mfccs_features_final
```

```
[19] import os
     import numpy as np
     from tqdm import tqdm

     ##iterate through every audio file and extract the features
     extracted_features=[]
     for index,row in tqdm(metadata.iterrows()):
       audio_file_name = os.path.join(os.path.abspath(audio_dataset_path),
       'fold'+str(row["fold"])+'/',str(row["slice_file_name"]))

       class_labels = row["class"]
       results_data = extract_features(audio_file_name)
       extracted_features.append([results_data, class_labels])
```

```
3555it [26:04,  2.85it/s]/usr/local/lib/python3.10/dist-packages/librosa/core/spectrum.py:257: Use
  warnings.warn(
8326it [58:53,  3.03it/s]/usr/local/lib/python3.10/dist-packages/librosa/core/spectrum.py:257: Use
  warnings.warn(
8329it [58:54,  3.27it/s]/usr/local/lib/python3.10/dist-packages/librosa/core/spectrum.py:257: Use
  warnings.warn(
8732it [1:01:34,  2.36it/s]
```

Fig 5. Code for function *extract_features()* and how it iterates over all audio through the metadata file

Inside the project, the MFCC method is applied in a function named *extract_features()* that will iterate through all the 8732 audio files in the dataset (see Fig 5 above). The extracted features and the corresponding class label the file is associated with then gets saved for the later training session.

```
[21] features_df = pd.DataFrame(extracted_features, columns = ['feature','class'])
     features_df.head()
```

	feature	class
0	[-217.35526, 70.22338, -130.38527, -53.282898,...	dog_bark
1	[-424.09818, 109.34077, -52.919525, 60.86475, ...	children_playing
2	[-458.79114, 121.38419, -46.520657, 52.00812, ...	children_playing
3	[-413.89984, 101.66371, -35.42945, 53.036354, ...	children_playing
4	[-446.60352, 113.68541, -52.402214, 60.302044,...	children_playing

Fig 6. Preview of results for MFFC-extracted features (converted to Data Frame from the first five audios in the dataset)

C. Model Creation

The construction of the model involves several stages. Initially, the model is instantiated, as illustrated in the code snippet provided and depicted in the model summary in Figure 7, according to the ANN model as explained beforehand.

This model is a linear stack of layers, where the output of one layer becomes the input for the next layer in a sequential manner. Rectified Linear Unit (ReLU) is also used as the activation function for all three layers since it introduces non-linearity to the model, allowing it to learn complex patterns that are especially needed for audio data. The first layer consists of 100 neurons, the second layer with 200 neurons, while the third layer contains 100 neurons. At the last layer, there are *num_labels* amount of neurons, where *num_labels* is the number of output classes, since the model outputs a probability distribution over the output classes. Softmax activation function is used at this final layer to convert the raw output scores into probabilities, making it suitable for multi-class classification problems.

```
[104] from tensorflow.keras.models import Sequential
      from tensorflow.keras.layers import Dense,Dropout,Activation,Flatten
      from tensorflow.keras.optimizers import Adam
      from sklearn import metrics
```

```
[105] num_labels = y.shape[1]
```

```
[106] model=Sequential()
      ###first layer
      model.add(Dense(100,input_shape=(40,)))
      model.add(Activation('relu'))
      model.add(Dropout(0.5))
      ###second layer
      model.add(Dense(200))
      model.add(Activation('relu'))
      model.add(Dropout(0.5))
      ###third layer
      model.add(Dense(100))
      model.add(Activation('relu'))
      model.add(Dropout(0.5))

      ###final layer
      model.add(Dense(num_labels))
      model.add(Activation('softmax'))
```

Fig 7. Code showing the first, second, third, and final layer

```
[107] model.summary()
      model.compile(loss='categorical_crossentropy',metrics=['accuracy'],optimizer='adam')
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 100)               4100

 activation_4 (Activation)   (None, 100)               0

 dropout_3 (Dropout)         (None, 100)               0

 dense_5 (Dense)             (None, 200)               20200

 activation_5 (Activation)   (None, 200)               0

 dropout_4 (Dropout)         (None, 200)               0

 dense_6 (Dense)             (None, 100)               20100

 activation_6 (Activation)   (None, 100)               0

 dropout_5 (Dropout)         (None, 100)               0

 dense_7 (Dense)             (None, 10)                1010

 activation_7 (Activation)   (None, 10)                0

=================================================================
Total params: 45410 (177.38 KB)
Trainable params: 45410 (177.38 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Fig 8. Overall model summary

Following this initialization phase, the model undergoes an extensive training regimen lasting for a duration of 00:01:46. This training session spans across 150 epochs, each epoch representing a complete pass through the entire training dataset. During these epochs, the model learns to adjust its parameters iteratively, gradually refining its ability to capture the underlying patterns and relationships present in the training data.

```
[116] from tensorflow.keras.callbacks import ModelCheckpoint
      from datetime import datetime

      num_epochs = 150
      num_batch_size = 32

      checkpointer = ModelCheckpoint(filepath='saved_models/audio_classification.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=[checkpointer], verbose=1)
      duration = datetime.now() - start
      print("Training completed in time: ", duration)
```

Fig 9. Code that executes the training session for the model

Through this prolonged training process, it is found that after model **evaluation**, the accuracy is at a rounded 77% (see Fig 10). Though it is certainly quite optimistic considering the common threshold of 70% on general models, the project aims to be able to classify the audios with better accuracy in the future. For now, to demonstrate the abilities of the model, there will be entirely new audios (that have never been 'fed' into the model as they are taken from various other external sources) used for this classification testing.

```
[117] test_accuracy = model.evaluate(X_test,y_test,verbose=0)
      print(test_accuracy[1])
```

```
0.7773325443267822
```

Fig 10. Preview of the accuracy after calling *model.evaluate()* and feeding the model with the testing data

IV. RESULTS AND DISCUSSIONS

Demonstrations of the model's capability in classifying new files of urban sounds will be able to help evaluate its accuracy qualitatively. The accuracy was found to be 77%, which is then expected that if four new audios (with prior knowledge as to which class they should correctly belong) are tested, about three can be assumed to be classified correctly according to its expected class.

Before immediately testing, two functions are created to help with pre-processing the audio with librosa library (extracting its feature with MFCC, a crucial step before predicting the class with the model) and displaying its final class output.

```
[109] import numpy as np
      import librosa

      def test_new_audio(test_file):
        test_audio, test_audio_sample_rate = librosa.load(test_file, res_type='kaiser_fast')
        mfccs_ftrs = librosa.feature.mfcc(y=test_audio, sr=test_audio_sample_rate, n_mfcc=40)
        mfccs_ftrs_final = np.mean(mfccs_ftrs.T, axis=0)

        mfccs_ftrs_final = mfccs_ftrs_final.reshape(1, -1)
        predicted_probs = model.predict(mfccs_ftrs_final)

        #note : convert probabilities to class labels, and take the class
        #with highest probability
        threshold = 0.5
        predicted_labels = (predicted_probs > threshold).astype(int)
        predicted_labels = np.argmax(predicted_probs, axis=1)

        return predicted_labels
```

```
[110] def print_classification_result(predicted_labels, actual_class):
        print("Predicted labels:")
        print(predicted_labels)

        #note : to convert predicted labels to class names
        prediction_class = labelencoder.inverse_transform(predicted_labels.flatten())
        print("Model-predicted class of the audio:")
        print(prediction_class)

        print("Actual class of the audio:")
        print(actual_class)
```

Fig 11. Code snippets of *test_new_audio()* function that handles pre-processing and model prediction as well as *print_classification_results()* for results and comparison output.

Upon completion of trying to predict the classes for each of the new audio files, the results are shown in the table below :

Audio Name	Predicted Label	Actual Class of Audio File	Predicted Class by Model
newaudio_dog_bark.wav	[3]	'dog_bark'	'dog_bark'
newaudio_vehicle_horn.wav	[1]	'car_horn'	'car_horn'
newaudio_construction_hammer.wav	[7]	'jackhammer'	'jackhammer'
newaudio_police_siren.wav	[2]	'siren'	'children_playing'

```
[111] print("Classification Results For New Audio Data EXAMPLE 1")
      print()

      audio1_test_file = '/content/drive/MyDrive/UrbanSound8K/newaudio_dog_bark.wav'
      actual_class = "dog_bark"

      print("File name of raw new audio : newaudio_dog_bark.wav")
      print()
      predicted_labels = test_new_audio(audio1_test_file)
      print_classification_result(predicted_labels,actual_class)
```

```
Classification Results For New Audio Data EXAMPLE 1

File name of raw new audio : newaudio_dog_bark.wav

1/1 [==============================] - 0s 82ms/step
Predicted labels:
[3]
Model-predicted class of the audio:
['dog_bark']
Actual class of the audio:
dog_bark
```

```
[112] print("Classification Results For New Audio Data EXAMPLE 2")
      print()

      audio2_test_file = '/content/drive/MyDrive/UrbanSound8K/newaudio_vehicle_horn.wav'
      actual_class = "car_horn"

      print("File name of raw new audio : newaudio_vehicle_horn.wav")
      print()
      predicted_labels = test_new_audio(audio2_test_file)
      print_classification_result(predicted_labels,actual_class)
```

```
Classification Results For New Audio Data EXAMPLE 2

File name of raw new audio : newaudio_vehicle_horn.wav

1/1 [==============================] - 0s 26ms/step
Predicted labels:
[1]
Model-predicted class of the audio:
['car_horn']
Actual class of the audio:
car_horn
```

```
[113] print("Classification Results For New Audio Data EXAMPLE 3")
      print()

      audio3_test_file = '/content/drive/MyDrive/UrbanSound8K/newaudio_construction_hammer.wav'
      actual_class = "jackhammer"

      print("File name of raw new audio : newaudio_construction_hammer.wav")
      print()
      predicted_labels = test_new_audio(audio3_test_file)
      print_classification_result(predicted_labels,actual_class)
```

```
Classification Results For New Audio Data EXAMPLE 3

File name of raw new audio : newaudio_construction_hammer.wav

1/1 [==============================] - 0s 26ms/step
Predicted labels:
[7]
Model-predicted class of the audio:
['jackhammer']
Actual class of the audio:
jackhammer
```

Fig 12. Code snippets for three new audio samples that are correctly classified by the model

```
[126] print("Classification Results For New Audio Data EXAMPLE 4")
      print()

      audio4_test_file = '/content/drive/MyDrive/UrbanSound8K/newaudio_police_siren.wav'
      actual_class = "siren"

      print("File name of raw new audio : newaudio_police_siren.wav")
      print()
      predicted_labels = test_new_audio(audio4_test_file)
      print_classification_result(predicted_labels,actual_class)

      Classification Results For New Audio Data EXAMPLE 4

      File name of raw new audio : newaudio_police_siren.wav

      1/1 [==============================] - 0s 32ms/step
      Predicted labels:
      [2]
      Model-predicted class of the audio:
      ['children_playing']
      Actual class of the audio:
      siren
```

Fig 13. Code snippet for the fourth audio sample that are incorrectly classified by the model

The four randomly sampled new audio files that have been tested and have their classes be predicted by the model yield a 75% overall accuracy, since three out of four was correctly classified while the last is not. There are many possible reasons for this, some plausible explanation are:

- Insufficient data : the model is trained on a limited dataset that may not adequately represent the variability present in real-world audio samples.
- Feature representation: features extracted from the audio may not capture all the relevant information necessary for classification, such as having chosen parameters (such as frame size, number of coefficients, etc.) not appropriate for the given task, which affect the model's performance.
- Model complexity: the model architecture may be too simple to capture the underlying complexity of the data, or perhaps overfit.
- Noise and Variability: the real-world audio data contains background noise or other sources of variability, which the model is not entirely robust to yet.

In conclusion, the model created for this project is able to classify urban sounds and therefore **reach the desired goal**. However, there **must be improvements** in its accuracy, possibly having an accuracy target of above 95% from the current accuracy of 77%. These can be further explored by altering some of the model hyperparameters such as learning rate, dropout rate, batch size, and others. In addition, a bigger dataset can also be used to improve the training data. To get the entire code notebook used for this project, the link here can be accessed : PATTERN RECOGNITION PROJECT.ipynb

V. REFERENCE

1. https://urbansounddataset.weebly.com/urbansound8k.html
2. https://en.wikipedia.org/wiki/Neural_network_(machine_learning)
3. https://cemse.kaust.edu.sa/cs/tags/artificial-neural-network
4. https://medium.com/@derutycsl/intuitive-understanding-of-mfccs-836d36a1f779
5. https://www.researchgate.net/figure/MFCC-feature-extraction-process_fig13_251719690
6. https://stackoverflow.com/
7. https://pixabay.com/
8. https://www.geeksforgeeks.org/what-does-the-output-of-model-predict-function-from-keras-mean/
9. https://www.geeksforgeeks.org/what-does-the-output-of-model-predict-function-from-keras-mean/