Vibration-Based Feedback Devices for Environmental Sound Recognition

Yüksel Çağrı Arslan Mustafa Eren Cen

May 25, 2025

1. Introduction

This report presents a detailed account of the implementation of a system that recognizes environmental sounds and delivers haptic feedback to assist people with hearing impairments. The primary goal is to classify key environmental audio events using machine learning and provide appropriate vibration and text-based alerts to users through a mobile platform.

2. Problem Statement and Motivation

People with hearing impairments face challenges in detecting environmental sounds such as sirens, alarms, or a baby crying. These are critical for safety and daily interaction.

This project aims to build a mobile-friendly system that listens to environmental sounds and notifies users through vibrations and visual alerts. The motivation is to improve inclusivity and awareness for individuals who cannot hear such important cues.

3. Dataset Preparation and Setup

To build the system, we utilized the ESC-50 dataset, which is available on Kaggle. The dataset contains 2,000 5-second audio clips spanning 50 environmental sound categories. For our purposes, we filtered a subset of 6 key categories that are highly relevant for hearing-impaired users:

`dog`, `thunderstorm`, `crying_baby`, `door_wood_knock`, `siren`, `car_horn`.

3.1 Library Imports and Environment Setup

First, we installed the necessary libraries and imported them in our environment:

```
import os, random, numpy as np, pandas as pd
import matplotlib.pyplot as plt, seaborn as sns
import librosa, librosa.display, noisereduce as nr
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

3.2 Downloading the Dataset

We used the Kaggle API to download the ESC-50 dataset. The steps included uploading the `kaggle.json` credential file, setting appropriate permissions, and running the following commands:

```
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d mmoreaux/environmental-sound-classification-50
!unzip -q environmental-sound-classification-50.zip
```

3.3 Loading Metadata and Filtering Categories

Once the audio and metadata files were extracted, we filtered the classes of interest. The metadata is stored in a CSV file that includes information such as filename, fold number, and category label.

```
meta = pd.read_csv('/content/esc50.csv')
selected_categories = [
    "dog", "thunderstorm", "crying_baby",
    "door_wood_knock", "siren", "car_horn"
]
filtered_meta =
    meta[meta["category"].isin(selected_categories)].reset_index(drop=True)
filtered_meta = filtered_meta.drop(columns=["esc10", "src_file", "take", "fold"])
```

This operation resulted in a reduced and focused dataset that will be used for training and evaluation. The dataset now includes only audio clips relevant to safety and daily life.

4. Exploratory Analysis and Signal Visualization

In this section, we explore the characteristics of the filtered ESC-50 dataset. This includes an analysis of clip durations, signal energy, and class distribution, as well as visual inspection via waveform and spectrogram plots.

4.1 Class Distribution

We verified that the selected categories are evenly distributed across the dataset. Each class contains approximately 40 samples.

4.2 Duration Analysis

We computed the duration of each audio clip in seconds. As expected, the ESC-50 dataset provides uniform-length clips, each lasting exactly 5 seconds.

4.3 RMS Energy Distribution

To evaluate signal strength across samples, we computed the Root Mean Square (RMS) energy. This distribution indicates that most clips are low-energy, with a few high-energy outliers.

4.4 Sample Waveforms and Spectrograms

We randomly sampled four clips from the dataset and visualized both their time-domain waveform and frequency-domain spectrograms. This helps illustrate the variability of sounds in both shape and energy spectrum.

4.5 Code Snippets for Analysis

```
# Plot audio durations
durations = [len(librosa.load(f"{DATA_PATH}/{f}", sr=None)[0])/sr for f in
    filtered_meta['filename']]
plt.hist(durations, bins=20)

# Compute and plot RMS energy
energies = [np.mean(librosa.feature.rms(y=librosa.load(f"{DATA_PATH}/{f}",
    sr=None)[0])) for f in filtered_meta['filename']]
sns.histplot(energies, bins=20)

# Visualize random samples
selected_files = random.sample(all_files, 4)
for file in selected_files:
    y, sr = librosa.load(os.path.join(DATA_PATH, file), sr=None)
    librosa.display.waveshow(y, sr=sr)
    D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
    librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='log')
```

5. Feature Extraction and Data Augmentation

To enhance the robustness of the classifier and improve generalization to real-world conditions, we applied multiple audio preprocessing and augmentation techniques before extracting features.

5.1 Audio Preprocessing

Each raw audio file undergoes the following operations:

1. **Silence trimming and noise reduction:** Silence and background noise are removed using `noisereduce`.
2. **Resampling:** All audio is resampled to a uniform sample rate of 44.1 kHz for consistency.

```
def adapt_audio(file_path, target_sr=44100):
    y, sr = librosa.load(file_path, sr=None)
    y = nr.reduce_noise(y=y, sr=sr)
    if sr != target_sr:
        y = librosa.resample(y, orig_sr=sr, target_sr=target_sr)
    return y, target_sr
```

5.2 Data Augmentation Techniques

To simulate real-world acoustic variability, we applied three augmentation strategies:

- **White noise injection** – simulates background interference.
- **Pitch shifting** – simulates pitch variation in sounds.
- **Time stretching** – alters playback speed.

```
def add_noise(y, noise_level=0.005):
    noise = np.random.randn(len(y))
    return y + noise_level * noise

def pitch_shift(y, sr, n_steps=2):
    return librosa.effects.pitch_shift(y=y, sr=sr, n_steps=n_steps)

def time_stretch(y, rate=1.1):
    return librosa.effects.time_stretch(y=y, rate=rate)
```

5.3 MFCC-Based Feature Extraction

We extracted 40-dimensional MFCC features along with their delta and delta-delta coefficients. For each audio file, four versions were used:

- Original audio
- Noisy version
- Pitch-shifted version

- Time-stretched version

Each version produced a feature vector by computing the mean across time for all 120 coefficients (40 + 40 + 40).

```
def extract_augmented_mfccs(file_path):
    y, sr = adapt_audio(file_path)
    features = []

    for version in [y, add_noise(y), pitch_shift(y, sr)]:
        mfcc = librosa.feature.mfcc(y=version, sr=sr, n_mfcc=40)
        delta = librosa.feature.delta(mfcc)
        delta2 = librosa.feature.delta(mfcc, order=2)
        combined = np.vstack([mfcc, delta, delta2])
        features.append(np.mean(combined.T, axis=0))

    try:
        y_stretch = time_stretch(y)
        mfcc = librosa.feature.mfcc(y=y_stretch, sr=sr, n_mfcc=40)
        delta = librosa.feature.delta(mfcc)
        delta2 = librosa.feature.delta(mfcc, order=2)
        combined = np.vstack([mfcc, delta, delta2])
        features.append(np.mean(combined.T, axis=0))
    except:
        pass

    return features
```

5.4 Dataset Preparation

For each audio file, features were extracted from original and augmented versions. All feature vectors were stacked, and corresponding labels were encoded using `LabelEncoder`. The dataset was split using stratified sampling for balanced class representation.

```
X_mfcc, y = [], []
for _, row in tqdm(filtered_meta.iterrows(), total=len(filtered_meta)):
    feats = extract_augmented_mfccs(os.path.join(DATA_PATH, row['filename']))
    for feat in feats:
        X_mfcc.append(feat)
        y.append(row['category'])

X_mfcc = np.array(X_mfcc)
y_encoded = LabelEncoder().fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(
    X_mfcc, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)
```

6. Model Training and Evaluation

To evaluate different classification strategies for environmental sound recognition, we trained and optimized four types of models:

- Random Forest Classifier (RF)
- Support Vector Machine (SVM)
- Multi-Layer Perceptron (MLP)
- Decision Tree Classifier (DT)

For each model, we performed randomized hyperparameter search using `RandomizedSearchCV` with 3-fold cross-validation. The models were trained on augmented MFCC features described in Section 5.

6.1 Hyperparameter Optimization Example

Below is the optimization code for the Random Forest classifier:

```
param_dist_rf = {
    'n_estimators': randint(20, 100),
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 4],
    'min_samples_leaf': [1, 2],
    'max_features': ['sqrt'],
    'bootstrap': [True, False]
}

random_search_rf = RandomizedSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_distributions=param_dist_rf,
    n_iter=10, cv=3,
    random_state=42, n_jobs=-1, verbose=2
)
random_search_rf.fit(X_train, y_train)
```

6.2 Best Parameters and Results

Random Forest: Best parameters: {`bootstrap=False, max_depth=20, max_features='sqrt', min_samples_leaf=2, min_samples_split=2, n_estimators=83`}

SVM: Best parameters: {`C=0.68, kernel='linear', gamma='auto'`}
Accuracy: **0.82**

MLP: Best parameters: {`activation='tanh', alpha=0.000625, hidden_layer_sizes=(64,), learning_rate='adaptive', solver='adam'`}

Table 1: Random Forest Classification Report

Class	Precision	Recall	F1-score
car_horn	0.94	0.91	0.92
crying_baby	0.91	0.97	0.94
dog	0.93	0.88	0.90
door_wood_knock	0.88	0.88	0.88
siren	0.94	0.91	0.92
thunderstorm	0.88	0.94	0.91
Accuracy		**0.91**	

Accuracy: **0.89**

Decision Tree: Best parameters: {`criterion='gini', max_depth=None, min_samples_leaf=1, min_samples_split=2, max_features=None`}
Accuracy: **0.68**

6.3 Summary of Results

- The **Random Forest Classifier** achieved the best performance with an accuracy of 91% and robust results across all classes.
- The **MLP Classifier** followed closely with 89% accuracy, suggesting that a simple neural network can also capture patterns well.
- The **SVM** showed moderate performance at 82%, which is consistent with its linear kernel assumption.
- The **Decision Tree** performed the weakest (68%), indicating overfitting or lack of generalization.

All evaluations used the `classification_report` function from scikit-learn to compute class-wise precision, recall, and F1-score.

7. Deep Learning with CNN and Log-Mel Spectrograms

In addition to classical machine learning classifiers, we trained a Convolutional Neural Network (CNN) on log-mel spectrogram representations of environmental sounds. Spectrogram-based models are better suited for capturing temporal and frequency patterns in audio.

7.1 Log-Mel Spectrogram Extraction

We augmented each audio file using the following transformations:

- Original audio

- Noise injection
- Pitch shift ($\pm$ 2 steps)
- Time stretching (faster and slower variants)

Each variant was then converted to a log-mel spectrogram of shape 128×216 with 128 mel bands and time axis fixed to 216 frames:

```
def extract_logmel_from_audio(y, sr, n_mels=128, hop_length=512, fix_len=216):
    mel = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=n_mels,
        hop_length=hop_length)
    log_mel = librosa.power_to_db(mel)
    log_mel = librosa.util.fix_length(log_mel, size=fix_len, axis=1)
    return log_mel[..., np.newaxis]
```

7.2 CNN Architecture

The model is composed of three convolutional layers followed by max pooling, a dropout layer to prevent overfitting, and two dense layers. The input shape is `(128, 216, 1)`:

```
cnn = Sequential([
    Input(shape=(128, 216, 1)),
    Conv2D(32, (3, 3), activation='relu'), MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'), MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'), MaxPooling2D((2, 2)),
    Flatten(), Dropout(0.3),
    Dense(128, activation='relu'),
    Dense(num_classes, activation='softmax')
])
```

7.3 Training Results

The model was trained over 20 epochs with batch size 32 and a 20% validation split. Training and validation accuracy rapidly converged after just a few epochs. Below is a snapshot of training performance:

- **Best validation accuracy:** 99.13%
- **Final test accuracy: 100%**

7.4 Final Evaluation on Test Set

7.5 Observations

- The CNN outperformed all classical classifiers tested earlier (Random Forest, SVM, etc.).

Table 2: CNN Classification Report on Test Set

Class	Precision	Recall	F1-score
car_horn	1.00	0.98	0.99
crying_baby	1.00	1.00	1.00
dog	1.00	1.00	1.00
door_wood_knock	1.00	1.00	1.00
siren	1.00	1.00	1.00
thunderstorm	0.98	1.00	0.99
Accuracy		**1.00 (100%)**	

- Data augmentation played a critical role in achieving generalization and high test accuracy.
- Overfitting was mitigated through dropout and a validation split.

8. Real-World Testing and Notification Mapping

To validate the model in real-world conditions, we used a custom test set composed of audio samples with filenames indicating their true labels. Each file was processed and classified by all trained models, including the CNN. In this phase, we simulated the application environment where predictions trigger haptic or visual feedback.

8.1 Notification Mapping

For user feedback, each sound category is mapped to a specific emoji-based message for visual notification. This also supports accessibility for users who may not be literate (e.g., children).

Table 3: Category-to-Feedback Mapping

Label	Notification Text
`dog`	Dog barking detected!
`thunderstorm`	Thunderstorm sound detected!
`door_wood_knock`	Door knocking sound detected!
`car_horn`	Car horn detected!
`crying_baby`	Baby crying detected!
`siren`	Siren sound detected!

8.2 Evaluation Logic

The following evaluation strategy was used:

- Extract features (MFCC or log-mel) from test samples.

- Predict top-1 and top-2 most likely classes.
- If prediction confidence ≥ 0.6, mark as confident.
- Compare predicted class with true label inferred from filename.
- Play back the audio sample and display the predicted label.

8.3 Results Summary

Table 4: Accuracy of Each Model on Real-World Test Set

Model	Accuracy	Comments
Random Forest	0.50 (9/18)	Tends to confuse dog and baby sounds.
SVM	0.50 (9/18)	Struggles with thunderstorm and siren.
MLP	0.39 (7/18)	Good on dog/baby, weak on thunder.
Decision Tree	0.33 (6/18)	Overfitted to training data.
CNN (Log-Mel)	**0.72 (13/18)**	Most robust and accurate on unseen data.

8.4 Observations

- CNN-based classification consistently yielded the highest accuracy in real-world scenarios.
- Confusion often occurred between dog barking and baby crying or siren due to overlapping frequency characteristics.
- Models trained only on ESC-50 may still struggle with audio recorded in less clean or noisier real-world environments.
- The emoji-based feedback system enhances accessibility, particularly for children or individuals with limited reading ability.

8.5 Playback Interaction and Confidence Thresholds

In the final prototype, each prediction also triggers audio playback with the associated label. If prediction confidence is below 60%, the system outputs both top-1 and top-2 guesses, as shown:

```
thunderstorm_1.wav:    Not confident
1.    Dog barking detected!  (0.59)
2.    Car horn detected!  (0.25)
```

9. Limitations

- The model is trained only on ESC-50 data, which might not cover all environmental conditions or sound types.
- Real-time processing performance on different mobile hardware has not been benchmarked.
- No formal user study has been conducted with hearing-impaired individuals to validate usefulness.

10. Conclusion and Future Work

We demonstrated a complete pipeline for environmental sound recognition using both classical ML and CNNs on augmented spectrograms.

Future Work includes:

- Deployment on Android/iOS with real-time audio input.
- Personalized vibration mapping based on user preferences.
- Extension to multilingual textual feedback and icon-based UIs.
- Testing with actual hearing-impaired users and collecting UX feedback.