Detailed Summary of the Audio Classification Project

The goal of this project was to build an *Audio Classification Model* capable of categorizing audio samples into predefined classes. The project followed a structured workflow, starting from data preparation to testing and evaluating the model. Here's a detailed summary of **everything we have done so far:**

1. Dataset Preparation:

Audio Data Source:

An audio dataset was used, which contained several labelled audio samples. Each sample belonged to a specific class, such as wav_arrayMic_F01, wav_headMic_FC03S01, etc.

- These labels represent different microphone setups or configurations.

Data Imbalance Handling:

- The dataset showed some classes having a significantly lower number of samples than others. To address this issue, SMOTE (Synthetic Minority Oversampling Technique) was applied. This method artificially increased the number of samples in underrepresented classes to balance the dataset.

Feature Extraction:

- Audio signals were pre processed to extract meaningful features for classification. Specifically:

- MFCCs (Mel-Frequency Cepstral Coefficients) were extracted from each audio sample. MFCCs are widely used in audio analysis tasks as they represent the frequency domain characteristics of the signal.

- The mean values of the MFCC features were computed to create a fixed-size feature vector for each audio sample.

2. Model Building:

Architecture:

- A neural network model was designed using TensorFlow/Keras to classify the audio samples. Key components of the architecture included:

- Input Layer: Accepts the feature vector (MFCCs) as input.

- Dense Layers: Fully connected layers with *ReLU* activation functions to capture patterns in the data.

- **Dropout:** Regularization technique used to prevent overfitting.

- **Output Layer**: A softmax layer to output probabilities for each class.

- Training:

- The model was trained on the processed audio features with corresponding labels.

-Training Results:

- Accuracy: 94%

- The model demonstrated high precision, recall, and F1-scores during training, indicating that it learned the patterns in the data effectively.

3. Validation

- Validation Dataset:

- A portion of the dataset was reserved for validation to assess the model's performance on unseen data.

- Validation Results:

- Accuracy: 86%

- While the validation accuracy was slightly lower than the training accuracy, it was still strong, indicating that the model generalized well to unseen data.

- Some classes, such as wav_headMic_FC03S01 and wav_headMic_FC03S02, exhibited lower performance, suggesting room for improvement.

4. Testing:

- After training and validation, the model was evaluated on an *independent test dataset* to assess its generalization capability.

- Test Results:

- Accuracy: 86%

- The test results aligned closely with the validation results, confirming that the model generalizes well.

- Some classes performed exceptionally well (e.g., wav_arrayMic_MC04S01 and wav_headMic_MC01S03 had F1-scores of 100%), while others showed room for improvement (e.g., wav_headMic_FC03S01 and wav_headMic_FC03S02 had F1-scores of 55% and 62%, respectively).

5. Model Saving:

- The trained model was saved as an HDF5 file (audio_classification_model.h5) for future use. This allows the model to be deployed or fine-tuned later.

6. Dynamic Input for Classification:

- We implemented a system to allow the model to classify audio samples dynamically. This involves:

- Audio Capture:

- Using a microphone to record live audio input from the user.

- Libraries like sounddevice were used for capturing audio.

- Feature Extraction:

- The recorded audio is preprocessed to extract MFCC features, just like during training.

- Prediction:

- The extracted features are passed to the trained model, which predicts the class of the audio.

- Interactive Loop:

- Users can continuously record audio and get predictions until they choose to exit.

7. Challenges Encountered:

- PortAudio Issue:

- While implementing dynamic input on Kaggle, an error was encountered due to the unavailability of the PortAudio library (required by the sounddevice library). Since system-level installations aren't supported on Kaggle, capturing live audio isn't feasible on this platform.

- As a workaround, we provided an alternative method to classify pre-recorded audio files uploaded to Kaggle.

- Class Imbalance:

- Some classes still showed lower performance despite applying SMOTE, indicating the need for additional data augmentation or rebalancing strategies.

8. Final Workflow for Dynamic Classification:

For users who want to classify audio dynamically:

1. On a Local Machine:

- Install necessary libraries (e.g., sounddevice and PortAudio).

- Use the provided script to record live audio, preprocess it, and classify it using the trained model.

2. On Kaggle:

- Upload pre-recorded .wav files to Kaggle.

- Use an alternative script to classify these files.

9. Future Work:

To improve the model further:

1. Enhance Low-Performing Classes:

- Apply advanced data augmentation techniques or collect more data for poorly performing classes.

2. Deploy the Model:

- Package the model and preprocessing pipeline for deployment, either as a web app or an API.

3. Integrate Speech Enhancement:

- Add a noise reduction or speech enhancement layer to improve the quality of audio input before classification.

4. Explore Advanced Architectures:

- Experiment with more sophisticated architectures like CNNs (Convolutional Neural Networks) or pre-trained models (e.g., Wav2Vec2) for audio classification.

10. Conclusion:

This project successfully built a robust audio classification model capable of classifying audio samples with high accuracy. While there are areas for improvement, the model is ready for deployment and can process both pre-recorded and live audio inputs for classification.