README

1 Introduction

This project is for audio classification and implements various models in Python. The primary focus is on using Mel-frequency cepstral coefficients (MFCCs) as features for machine learning and deep learning models. The following scripts are included in the project:

- `mlp.py`: Creates and trains a multi-layer perceptron (MLP) for audio classification.
- `cnn.py`: Implements a Convolutional Neural Network (CNN) for audio classification.
- `pretrained.py`: Uses a pre-trained VGG16 model with additional layers for audio classification.
- `plots.py`: Generates visualizations for audio data, including waveforms and spectrograms.

2 Scripts Description

2.1 MLP Model

The `mlp.py` script creates and trains an MLP model to classify audio recordings using MFCC features. The process involves:

- Loading MFCC features and labels from a CSV file.
- Splitting data into training, validation, and test sets with stratification.
- Applying standardization to features.
- Training the MLP model with a single hidden layer, using dropout for regularization.
- Evaluating the model using precision, recall, F1-score, and overall accuracy.

The script uses the Adam optimizer for training, with a learning rate and weight decay for regularization. Training and validation accuracy and loss are plotted after training.

2.2 CNN Model

The `cnn.py` script defines and trains a CNN model for audio classification using MFCC features. Key steps include:

- Using `librosa` to extract MFCC features and convert them to decibels (dB).
- Creating a custom dataset class to load and preprocess audio files during training.
- Defining a CNN architecture with convolutional layers, pooling layers, and fully connected layers.
- Using dropout for regularization and softmax for multi-class classification.
- Implementing a training and evaluation loop with PyTorch Lightning.

The script also handles data splitting, standardization, and other preprocessing steps.

2.3 Pre-Trained VGG16 Model

The `pretrained.py` script implements a system for audio classification using a pre-trained VGG16 model. The process includes:

- Loading and preprocessing audio files, similar to the CNN model.
- Utilizing a pre-trained VGG16 model with additional convolutional and fully connected layers.
- Using PyTorch Lightning for training and validation.
- Applying dropout for regularization and Adam optimizer for training.
- Overriding methods for custom testing and evaluation.

The script focuses on reusing pre-trained model weights and fine-tuning them for audio classification tasks.

2.4 Plots and Visualizations

The `plots.py` script generates various visualizations for audio data, including waveforms and spectrograms. Features include:

- Loading audio files from a specified directory.
- Calculating Short-Time Fourier Transform (STFT) to represent audio in the time-frequency domain.
- Generating linear, log, and Mel spectrograms for visualization.
- Plotting waveforms (amplitude over time) and spectrum analysis.

The script provides options for customizing and saving the generated figures.

3 Hyper-Parameters and Kaggle Submissions

This project uses various hyper-parameters for training and validation, including:

- **Learning Rate**: A range of values to control weight updates.
- **Momentum**: For maintaining training direction.
- **Dropout Rate**: To prevent overfitting.
- **Batch Size**: To manage memory and training stability.
- **Training, Validation, and Test Ratios**: To control data splits.

The project also includes code for Kaggle submissions. An example of Kaggle submission generation is shown below:

```
if PREDICT_KAGGLE_DATASET:
    df_kaggle = pd.read_csv(github + os.sep + f"kaggle_mfcc_{num_mfcc}.csv")
    X_kaggle = StandardScaler().fit_transform(df_kaggle.iloc[:, 0:])

    kaggle_dataset = TensorDataset(torch.tensor(X_kaggle, dtype=torch.float32))

    with torch.no_grad():
        numbered_preds = model(kaggle_dataset.tensors[0]).argmax(dim=1).tolist()
        predictions = unique_labels[numbered_preds]

    kaggle_submission = pd.DataFrame()
    kaggle_submission.insert(0, "id", files_in_test_dir)
    kaggle_submission.insert(1, "class", predictions)
    kaggle_submission.to_csv(save_kaggle_submission_as, index=False)
```

This code snippet demonstrates how to load Kaggle data, make predictions, and build a submission file.

4 Conclusion

The project provides a comprehensive framework for audio classification using different models and data processing techniques. Further improvements and optimizations could be made by exploring hyper-parameter tuning, early stopping, and other advanced training strategies.