Music genre Classifier

Major Project submitted to

Indian Institute of Information and Technology, Manipur

textit in partial fulfilment of the requirements for

The degree of

Bachelor of Technology

from

Department of Computer Science and Engineering

by

Anjali

under

Dr. Kabita Thaoroijam

Computer Science and Engineering

Indian Institute of Information and Technology, Manipur

Imphal, Manipur, 795002

November 2023

Certificate

This is to certify that the project report entitled“Music genre Classifer ” submitted to Department of Computer Science & Engineering, Indian Institute of Information Technology Senapati, Manipur in partial fulfillment for the award of the degree of Bachelor of Technology in Computer Science Engineering is a record of bonafide work carried out by Anjali bearing roll number 20010101.

Dr. Kabita Thaoroijam

Supervisor

Professor, Department of Computer Science Engineering

Indian Institute of Information Technology Senapati, Manipur

Date:

Declaration

I, Anjali, registered as undergraduate scholar, bearing Registration Number "20010101" for the Bachelor of TechnologyProgramme under the Computer Science and Engineering Department of Indian Institute of Information and Technology, Manipur do hereby declare that I have completed the requirements as per mentioned by the university for project submission.

I do hereby declare that the project submitted is original and is the outcome of the independent investigations /research carried out by me and contains no plagiarism. The work is leading to the discovery of new techniques. This work has not been submitted by any other University or Body in quest of a degree, diploma or any other kind of academic award.

I do hereby further declare that the text, diagrams or any other material taken from other sources (including but not limited to books, journals and web) have been acknowledged, referred and cited to the best of my knowledge and understanding.

Date:

Signature of Student

Anjali

20010101

Department of Computer Science and Engineering

IIIT Senapati, Manipur

Abstract

The project Music genre Classifier focuses on the development and evaluation of music genre classification models using machine learning techniques. Two distinct models, a Multi-Layer Perceptron (MLP) and a Convolutional Neural Network (CNN), are implemented and trained on the GTZAN dataset, a benchmark in the field. The MLP, with its fully connected architecture, demonstrates commendable accuracy in classifying music genres. In contrast, the CNN, designed to capture spatial hierarchies using Mel-frequency cepstral coefficients (MFCCs), showcases enhanced performance in recognizing complex patterns inherent in music genres.

The study recognizes the limitations of the GTZAN dataset, such as its relatively small size and limited genre diversity, and discusses their potential impact on model generalization. Future work is outlined, suggesting avenues for dataset expansion, integration of additional features, exploration of contemporary music styles, evaluation on larger datasets, and enhancement of model robustness to audio variability.

By addressing these considerations, the project aims to contribute to the advancement of music classification models, fostering improved performance across a wider range of genres and production styles. The findings pave the way for more accurate and versatile applications in music analysis, acknowledging the evolving landscape of musical expression.

Acknowledgements

I would like to express my special thanks of gratitude to my supervisor Dr. Kabita Thaoroijam, who gave me the opportunity to do this wonderful project on creating this Deep Learning project to make a "Music genre Classifier", who also helped me and guided me a lot in completing this Project . I came to know about so many new things and I am really very thankful to them.

-Anjali

Contents

List of Figures

Chapter 1

Introduction

The Music Genre Classifier is designed as a powerful tool for automatically categorizing music audio into one of the ten genres: blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, and rock. The primary motivation behind the development of this classifier is to enhance the user experience in exploring and organizing vast music libraries. Understanding the genre of a piece of music is valuable for personalized recommendations, playlist creation, and efficient music discovery.

1.1 Introduction

Music genre classification is a computational task that employs machine learning and deep learning algorithms to automatically assign predefined musical categories or genres to audio tracks. The objective is to develop models capable of recognizing patterns and features within the audio signals and its MFCCs, enabling the categorization of music into the following genres genres: blues, classical, country, disco, hiphop, jazz, metal, pop, reggae and rock.
The process involves training the model on the labeled dataset: GTZAN Dataset - Music Genre Classification, where each audio sample is associated with its corresponding genre label, and subsequently using the trained model to classify new, unseen music based on learned patterns. This model finds applications in music recommendation systems, content organization, and enhancing user experiences in digital music platforms.

The primary goal of our project is to automatically categorize music into distinct genres based on the inherent patterns and features within the audio signal like MFCCs leveraging advanced machine learning techniques, specifically multilayer neural netwoks and Convolutional Neural Networks (CNNs).

1.1.0.1 Features

- **Supervised Learning:** Supervised learning is a type of machine learning where the algorithm is trained on a labeled dataset, which means that each input in the training data is associated with the corresponding correct output. The goal of supervised learning is to learn a mapping from inputs to outputs so that the algorithm can make accurate predictions or decisions when given new, unseen data.
  Music genre classification is typically approached as a supervised learning problem, where the model is trained on a labeled dataset containing audio samples annotated with their respective genres.

- **MFCC:** MFCC stands for Mel Frequency Cepstral Coefficients. It is a feature widely used in the field of audio signal processing and speech recognition. MFCCs are coefficients that collectively represent the short-term power spectrum of a sound signal.MFCCs for all the audio data is stored and saved for working on the audio calssification

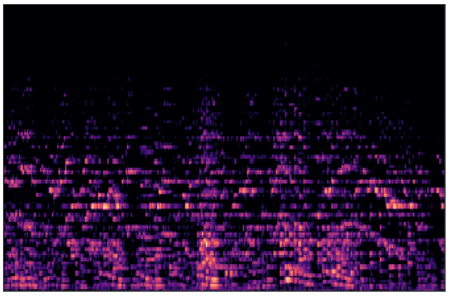


FIGURE 1.1: MFCC of a data model: from Dataset

- **Multi-Layer Perceptron:** Using a Multi-Layer Perceptron (MLP) for audio data involves leveraging the architecture and learning capabilities of MLPs to process and analyze audio features.

- Architecture:MLPs consist of an input layer, one or more hidden layers, and an output layer.Neurons in each layer are connected to neurons in the subsequent layer, forming a network of interconnected nodes.

- Activation Function:Neurons in an MLP use an activation function to introduce non-linearity into the model, allowing it to learn and represent complex relationships in the data.Common activation functions include the Rectified Linear Unit (ReLU) for hidden layers and softmax for the output layer in classification tasks.

- Training Algorithm: MLPs are trained using the backpropagation algorithm, which adjusts the weights of the connections based on the error between predicted and actual outputs.Optimization algorithms like stochastic gradient descent are often employed to minimize the loss function during training.

- Universal Approximator:The universal approximation theorem states that an MLP with a single hidden layer containing a sufficient number of neurons can approximate any continuous function.This property underscores the expressive power of MLPs in capturing intricate mappings between input and output data.

- Overfitting and Regularization: MLPs are susceptible to overfitting, where the model performs well on the training data but struggles with unseen data.Regularization techniques, such as dropout and L2 regularization, are commonly used to mitigate overfitting and enhance model generalization.

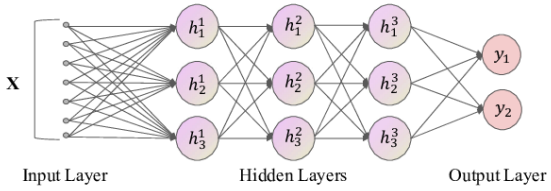


FIGURE 1.2: MultiNeural Perceptron, source: Google image

- **Convolutional Neural Networks:** Using Convolutional Neural Networks (CNNs) for audio data involves adapting the architecture and principles of CNNs to effectively process and analyze the temporal and spectral features of audio signals. Here's a

breakdown of the key considerations when applying CNNs to audio data, specifically for tasks like music genre classification:

- Data Representation: Audio data is typically represented as a spectrogram, a 2D visual representation of the audio signal's frequency content over time. The spectrogram serves as the input to the CNN, allowing the network to capture both temporal and spectral features.

- Convolutional Layers:The core of a CNN for audio data consists of convolutional layers. These layers use filters to convolve over the input spectrogram, capturing local patterns.Convolutional operations are effective in detecting hierarchical features, such as pitch variations and harmonic structures.

- Pooling Layers: Max pooling layers are commonly used to downsample the spatial dimensions of the input, reducing the computational load and extracting the most salient features.Pooling layers help make the model translation-invariant, allowing it to recognize patterns regardless of their exact position in the input.

- Architecture: The architecture of a CNN for audio data may include multiple convolutional layers followed by pooling layers. This hierarchical structure enables the network to learn increasingly complex representations.Batch normalization layers are often used to stabilize and accelerate the training process.

- Flattening and Dense Layers:After convolutional and pooling layers, the output is flattened to a 1D vector.Dense (fully connected) layers are then added to perform high-level reasoning and classification based on the learned features.

- Activation Functions: Common activation functions include Rectified Linear Unit (ReLU) in the convolutional layers and softmax in the output layer for multi-class classification tasks, such as music genre classification.

- Training and Optimization: CNNs for audio data are trained using supervised learning with labeled datasets.Optimization algorithms like Adam or RMSprop are employed to update the model's parameters based on the computed gradients.

– Hyperparameter Tuning: Parameters such as the learning rate, batch size, and the number of filters in each convolutional layer are tuned to optimize the model's performance.

– Data Augmentation: To enhance model robustness, data augmentation techniques, such as time stretching or pitch shifting, may be applied to artificially increase the diversity of the training dataset.

– Evaluation and Testing: The model is evaluated on a validation set during training to monitor its performance and prevent overfitting.Evaluation on a separate test set provides insights into the model's ability to generalize to unseen audio data.

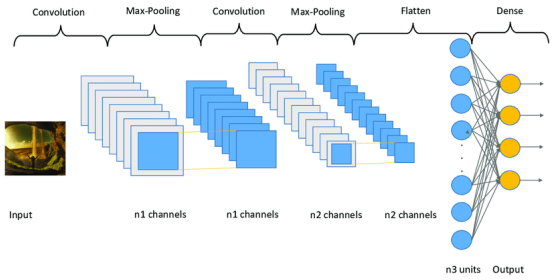


FIGURE 1.3: Convolutional Neural Network, source: Google image

1.2 Existing System

– **MusicCNN:** The MusicCNN model focuses on leveraging the power of CNNs for audio-based music classification, particularly when dealing with limited datasets.
Architecture: Convolutional Neural Network (CNN)

– **DeepMusicGenre:**DeepMusicGenre is a research project that explores deep learning techniques for music genre classification. It uses convolutional neural networks (CNNs) to analyze the spectrogram of audio files and achieved competitive results on benchmark datasets.
Architecture: Convolutional Neural Network (CNN)

– **Music Genre Classification with LSTM:** This model explores the use of LSTMs for automatic music tagging, showcasing the effectiveness of recurrent

neural networks in capturing temporal dependencies in audio sequences.
Architecture: Long Short-Term Memory network (LSTM)

- **Hybrid CNN-LSTM Models:** The hybrid model integrates the strengths of both CNNs and LSTMs, aiming to achieve enhanced performance in music genre classification tasks, especially when dealing with large datasets.
  Architecture: Combination of Convolutional Neural Network (CNN) and Long Short-Term Memory network (LSTM)
- **MagnaTagATune Dataset:** The MagnaTagATune dataset serves as a valuable resource for music genre classification tasks. Various neural network models, particularly those employing CNNs, have been explored to categorize music genres within this dataset.
  Architecture: Various architectures including CNNs

1.3 Motivation

The motivation behind undertaking the Music Genre Classification Project stems from the desire to address several key aspects within the realm of music information retrieval and user experience.
Here are the primary motivations guiding our project:

- **Enhancing Music Recommendation Systems:**Music streaming platforms often rely on accurate genre classification to enhance their recommendation algorithms. By implementing advanced machine learning models, we aim to contribute to the improvement of personalized music recommendations, providing users with a more tailored and enjoyable listening experience.

- **Exploring Deep Learning Architectures:**The project serves as an exploration into the capabilities of Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs) in the context of music genre classification. Understanding how these sophisticated neural architectures perform in comparison to traditional methods contributes to the ongoing evolution of machine learning techniques in music analysis.

- **Addressing Subjectivity in Genre Classification:**Music genres can be highly subjective and can vary across different cultures and individual preferences. By leveraging machine learning models, we aim to develop systems

that can capture and learn nuanced genre characteristics, potentially offering a more objective and data-driven approach to genre classification.

- **Enabling User-Centric Experiences:**Accurate genre classification lays the foundation for more user-centric features in music applications. This includes creating smart playlists, organizing radio stations, and tailoring content recommendations based on users' diverse musical tastes.

- **Practical Application in Industry:**The outcomes of our project have practical applications in the music industry, from improving music streaming services to aiding in the creation of smart music players. The integration of accurate genre classification can positively impact how users discover and engage with music.

In summary, our motivation is rooted in the desire to harness the power of advanced machine learning models to tackle challenges in music genre classification, with the ultimate goal of enhancing user experiences and contributing to the ongoing development of intelligent music systems.

1.4 Intented Use

- **Music Library Management:** The primary use of the Music Genre Classifier is to assist users in managing their music libraries more efficiently. Users can categorize their songs based on genre automatically, saving time and effort.

- **Increased Efficiency:** The custom SVM classifier function was designed to be more efficient than the SVC classifier in Scikit-learn library. It achieves this by using a simplified implementation of the SMO algorithm and avoiding the overhead of the Scikit-learn library.

- **Playlist Generation:** The classifier facilitates the creation of thematic playlists by automatically sorting songs into genres. Users can easily curate playlists tailored to specific moods, occasions, or activities.

- **Personalized Recommendations:**By understanding the genre preferences of users, the classifier contributes to personalized music recommendations. Users receive suggestions that align with their tastes, fostering a more engaging and enjoyable listening experience.

- **Music Discovery:** The classifier serves as a tool for music discovery, helping users explore genres they may not be familiar with. This feature encourages a broader musical experience by introducing users to new and diverse genres.

Chapter 2

System Design

The Music Genre Classifier was implemented leveraging two models, including Multilayer Neural Networks and Convolutional Neural Networks (CNNs), providing a comprehensive approach to music genre classification.It is trained on the "GTZAN Dataset - Music Genre Classification" dataset.

2.1 System Analysis

Firstly the user will take an audio/music file in the .wav or .mp3 format and the model will output the result as one of the music genres.

2.1.1 Process Flow

The process flow for the design of the models is as follows:

- **Data Preprocessing:** Each audio file is segmented into 10 parts to facilitate effective feature extraction.Audio features, specifically Mel-frequency cepstral coefficients (MFCCs), are extracted using the Librosa library, offering a robust foundation for music analysis.Genre labels are encoded numerically, ranging from 0 to 9, providing a quantitative representation of music genres.The data, including the "mapping," "labels," and "MFCCs," is organized into a dictionary for efficient handling.The dataset undergoes a split into training and testing sets,

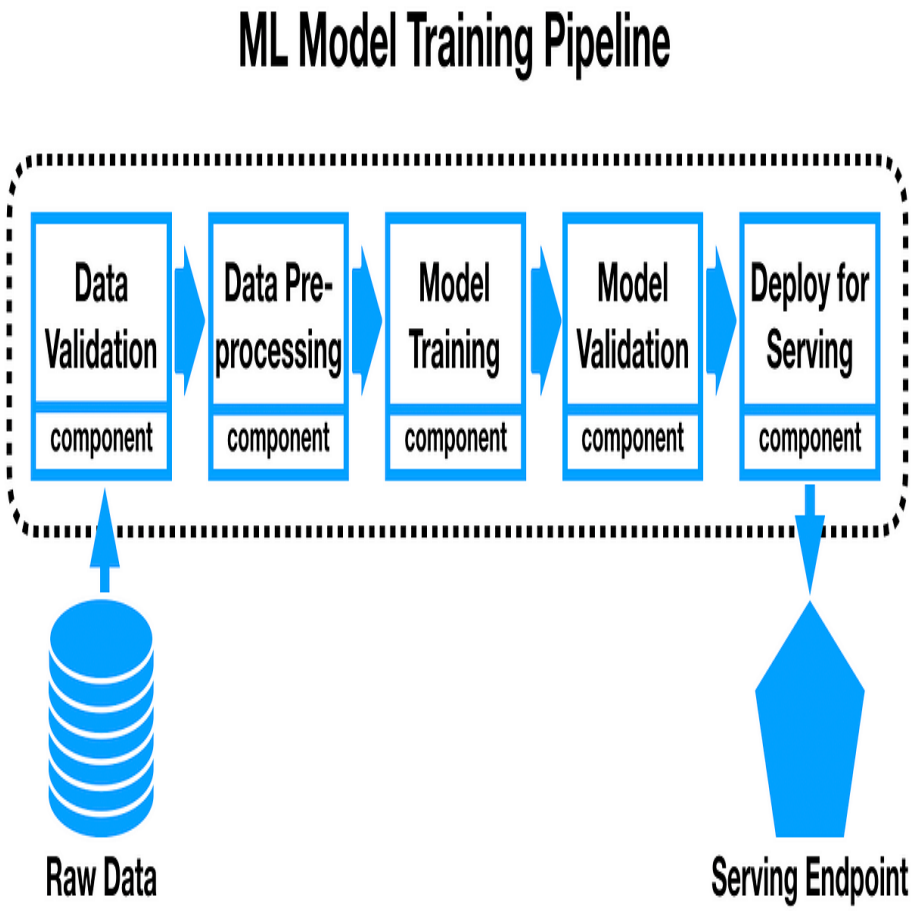


FIGURE 2.1: Process Flow Diagram, source: Google image

facilitated by the train-test split functionality from scikit-learn.The processed data is saved in a JSON file to be used for future use.It contains the mapping, labels and the MFCCs.

– **Model Architecture:**The various models are designed as follows:
MLP model: The Multi-Layer Perceptron (MLP) model is structured as a Sequential model.It comprises an input layer, three hidden layers, and an output layer.The hidden layers incorporate 512, 256, and 64 neurons, employing the Rectified Linear Unit (ReLU) activation function.

CNN model: TThe Convolutional Neural Network (CNN) model, implemented with Keras, involves:

  * Three convolutional layers with max-pooling and batch normalization for effective feature extraction.
  * A dense layer with ReLU activation and dropout to capture spatial hierarchies.
  * A final layer with softmax activation, facilitating the classification of music genres.

- **Preventing Overfitting:** To mitigate the risk of overfitting, the model incorporates Dropout layers and L2 regularization.
  * Dropout layers introduce randomness during training, reducing the dependency on specific neurons.
  * L2 regularization imposes penalties on large weight magnitudes, promoting generalization.
- **Model Compilation:** During the compilation phase, the model's architecture is linked with the training process by specifying essential elements such as the loss function and optimizer. The choice of the categorical crossentropy loss function is well-suited for classification tasks where each instance belongs to a single category. It quantifies the disparity between predicted and actual class probabilities.It uses the Adam optimizer,for its adaptive learning rate capabilities.The evaluation metric, accuracy, is strategically selected to align with the overarching goal of the project – accurately classifying music genres.

- **Model Training:** The heart of the model development process lies in the training phase, where the model learns to recognize patterns and relationships within the training dataset.The fit method is employed for training, enabling the model to iteratively update its parameters based on the training data.Crucial training parameters, including batch size and the number of epochs, are meticulously configured through experimentation. Batch size determines the number of training samples utilized in each iteration, while the number of epochs denotes the total number of training iterations. Finding an optimal balance for these parameters is a delicate yet essential task to ensure effective learning without overfitting or underfitting.

– **Model Evaluation:** The test set serves as a critical benchmark, representing data that the model has not encountered during training. This evaluation gauges the model's generalization performance, providing insights into its ability to extrapolate learned patterns to unseen instances.

2.2 System Requirements

2.2.1 Hardware requirements

To ensure optimal performance and efficient execution of the algorithm, specific hardware requirements must be met. The essential components include the Central Processing Unit (CPU), memory, and Graphics Processing Unit (GPU) for accelerated computation. The detailed specifications are as follows:

– **CPU:** The algorithm, being computationally intensive, may benefit from a powerful CPU, especially when processing large audio datasets or real-time audio streams.A multicore processor with higher clock speeds and efficient parallel processing capabilities is recommended to enhance the speed and efficiency of the algorithm.

– **Memory:** Sufficient memory space is crucial to accommodate the input dataset, especially considering the dataset size is 1.2 GB.
Ensure that the system has ample Random Access Memory (RAM) to handle the loading and manipulation of audio data during the preprocessing and model training stages.

– **GPU:** While not mandatory, GPU acceleration can significantly boost the algorithm's performance, particularly in scenarios involving complex neural network computations.For users seeking GPU acceleration in Python, it is necessary to have a compatible NVIDIA GPU.Additionally, the installation of the CUDA Toolkit and cuDNN (CUDA Deep Neural Network library) is required.

These tools enhance the TensorFlow library's ability to leverage GPU capabilities, resulting in faster and more efficient computation during the training and evaluation phases.

These hardware requirements are outlined to ensure that the algorithm can effectively handle the computational demands associated with music genre classification. Adhering to these specifications contributes to a seamless and high-performance execution, enabling users to harness the full potential of the algorithm for accurate and timely genre predictions.

2.2.2 Software requirements

To deploy and run the algorithm successfully, certain software requirements must be satisfied. The key components encompass operating system compatibility, integrated development environment (IDE), programming language, and essential libraries. The detailed specifications are as follows:

- **Operating system:** The algorithm can be run on various operating systems such as :
  - Windows 7 or later
  - Ubuntu 16.04 or later
  - macOS 10.12.6 (Sierra) or later (Note: GPU support is not required for execution)
- **Integrated Development Environment (IDE):** he algorithm can be executed within various integrated development environments. Recommended IDEs include:
  - PyCharm
  - Visual Studio Code (VSCode)
  - Any other environment that supports Python.
- **Programming language:** The Python programming language is a prerequisite for running the model effectively.
  It is imperative that Python is installed on the system, and the version should fall within the range of Python 3.8–3.11 for TensorFlow compatibility

- **Libraries:** The following libraries are essential for different aspects of the algorithm:
  - sklearn: Required for various data preprocessing tasks.
  - librosa: Used for loading audio data and extracting relevant features.
  - tensorflow: Fundamental for creating and training the model.
  - numpy: Essential for matrix calculations and numerical operations.
  - matplotlib: Necessary for visualizing and plotting the model's results.

  It is imperative that these libraries are installed on the system to ensure seamless execution and functionality of the algorithm. Installation can typically be performed using package managers like pip or conda.

In conclusion, running the classifier requires a computer with sufficient processing power, memory, and software libraries.

2.3 System/Software used in this model

The Music genre classifier model is implemented using Python programming language. The following software and libraries were used in the implementation:

- **PyCharm (Integrated Development Environment):** PyCharm serves as the primary IDE for developing and managing the codebase of the Music Genre Classifier. It provides a user-friendly interface with features such as code highlighting, debugging, and project organization, enhancing the development workflow.

- **TensorFlow:**TensorFlow is a critical component for building and training the machine learning models used in the Music Genre Classifier. It offers a comprehensive framework for designing neural network architectures, facilitating the implementation and optimization of models for music genre classification.

- **Librosa:**Librosa is employed for audio processing tasks within the system. It allows the extraction of essential features, such as Mel-frequency cepstral coefficients (MFCCs), enabling the representation of audio data in a format suitable

for training the classifier.

- **NumPy:** NumPy is utilized for numerical operations and efficient array manipulation. It plays a crucial role in handling data structures and computations, providing a foundation for various mathematical operations involved in the development of the classifier.

- **Matplotlib:** Matplotlib is used for data visualization, aiding in the analysis and interpretation of model performance. It allows the creation of informative plots and graphs, enabling developers to gain insights into the behavior of the classifier during training and evaluation.

- **Scikit-learn:** scikit-learn plays a pivotal role in the Music Genre Classifier, providing a versatile machine learning library for various tasks. Sklearn library functions were used in the model for train and test splitting and preprocessing.

- **TensorFlow:** TensorFlow is an open-source machine learning framework developed by the Google Brain team. It provides a comprehensive set of tools and libraries for building and deploying machine learning models. Here's an overview of key aspects of TensorFlow:

  Overall, the custom Classifier was implemented using a combination of popular software and libraries for data processing, visualization, and machine learning.

Chapter 3

Implementation

The implementation leverages Deep Learning (DL) techniques, utilizing libraries such as TensorFlow, librosa and matplotlib. This choice harnesses the power of neural networks for efficient and effective processing in the context of the specific task, contributing to the model's capability for music genre classification.

3.1 Dataset

The GTZAN dataset is a widely used benchmark dataset for music genre classification tasks. It is a collection of audio clips spanning various genres, making it valuable for evaluating and training machine learning models in the field of music genre classification.
Content of the dataset:

- **genres original:** A collection of 10 genres with 100 audio files each, all having a length of 30 seconds.The dataset covers ten music genres: blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock.
- **images original:** A visual representation for each audio file. Here we have the Mel Spectrograms of the audio data to use networks like CNNs on the data.
- **Two CSV files:** Containing features of the audio files. One file has for each song (30 seconds long) a mean and variance computed over multiple features that can be extracted from an audio file. The other file has the same structure, but the songs were split before into 3 seconds audio files.

3.2 Data Preprocessing

Data preprocessing for our model has been performed in the "extract_data.py file.

Here we initialized a dictionary called data to store 'mapping', 'labels', and 'MFCCs'.It calculates parameters like 'samples_per_segment' and 'num_mfcc_vectors_per_segment' based on the provided constants.
It iterates through sub-folders in the dataset, assuming each sub-folder represents a genre.
For each audio file in a genre, it loads the file using librosa.load.
It processes the audio file in segments, calculates the start and finish samples for each segment, and extracts MFCCs using librosa.feature.mfcc. Each audio file is divided into 10 samples, which later are stored and used independently.
The extracted MFCCs are stored in the data dictionary along with their corresponding labels (genre index) for each segment.
Finally, the data dictionary is saved to a JSON file specified by json10_path.

FIGURE 3.1: The load data function code, source: Project code

3.3 Preventing Overfitting

3.3.0.1 Dropout Method

– Objective: Dropout is a technique designed to reduce the interdependence of neurons in a neural network by randomly dropping (i.e., setting to zero) a fraction of them during training.

– Mechanism: During each training iteration, a random subset of neurons is "dropped out" with a certain probability. This forces the network to learn robust and redundant representations, as it cannot rely on specific neurons for every input.

– Effect: Dropout acts as a form of ensemble learning within a single model, preventing co-adaptation of neurons and improving generalization. It helps prevent overfitting by introducing a level of stochasticity during training, making the model more resilient to variations in the input data.

3.3.0.2 L2 Regularization

– Objective: L2 regularization, also known as weight decay, aims to prevent the model from becoming too reliant on a small number of features by penalizing large weights.

– Mechanism: It adds a regularization term to the loss function, proportional to the squared magnitude of the weights. This encourages the optimization process to prioritize smaller weight values.

– Effect: By discouraging the dominance of a few features, L2 regularization helps create models that generalize well to unseen data. It contributes to smoother decision boundaries and mitigates the risk of capturing noise in the training

data.

3.4 Multi-layer Neural Network

A Multilayer Perceptron (MLP) is a type of artificial neural network designed for supervised learning. It consists of multiple layers of nodes, including an input layer, one or more hidden layers, and an output layer.

3.4.1 Import Libraries

The model implementation begins by importing necessary libraries, including json for data loading, numpy for numerical operations, scikit-learn for dataset splitting, and TensorFlow's Keras for building and training the neural network.

3.4.2 Load data

The 'load_data' function reads MFCCs and labels from the 'data_10.json' JSON file, converting them into numpy arrays.It returns the X and y values, i.e, input audios and output labels respectively.

3.4.3 Split the data into train-test datasets

The data is split into training and testing sets using scikit-learn's train_test_split function.Here we have split the data such that 70% of the data is taken for training the model while the rest 30% is stored for testing the model accuracy and performance.

3.4.4 Built the model architecture

The neural network architecture is defined using Keras, consisting of a Flatten layer for input, followed by three dense hidden layers with varying units and ReLU activation functions. The output layer utilizes softmax activation and 10 neurons for multi-class classification.The number of neurons for the hidden layers are 512, 256 and 64 respectively.

To prevent overfitting of data in the model we have used L2 regularization with regularization strength of 0.001 and Dropout with probability of 0.3.

3.4.5 Compile the network

The model is compiled with an Adam optimizer, sparse categorical crossentropy loss, and accuracy as the evaluation metric. The learning rate for the Adam optimiser is taken as 0.0001.
A summary of the model architecture is printed, providing insights into the layers, parameters, and connections.

3.4.6 Train the network

The model is trained using the training set, with validation performed on the testing set using the fit method of the tensorflow's keras.
Batching, to calculate the gradient of the dataset, is done on a batch size of 32.
The number of epochs is 50,i.e, the model is trained iteratively on the training data 50 times.

3.5 Convolution Neural Network

CNNs are a class of deep neural networks, particularly effective for image recognition and classification.They can perform better than multilayer perceptrons while using lesser parameters than a dense layer.

The model implementation begins by importing necessary libraries, including json for data loading, numpy for numerical operations, scikit-learn for dataset splitting, and TensorFlow's Keras for building and training the neural network.

3.5.1 Load data

The 'load_data' function reads MFCCs and labels from the 'data_10.json' JSON file, converting them into numpy arrays.It returns the X and y values, i.e, input audios and output labels respectively.

3.5.2 Split data into train, test and validation datasets

Here the dataset is divided into three parts: train set, test set and validation set.The validation set is required for hyperparameter tuning, as it acts as a proxy for unseen data.
The prepare_datasets function is used to achieve it with the help of sklearn library's

train_test_split and then adding one more dimension to the result for 'depth' in the dataset.

3.5.3 Build the model architecture

The build_model function creates a CNN model using Keras.
The model has three convolutional layers each consisting of a 2D convolutional layer, max pooling layer and batch normalization layer.
Each 2D convolutional consists of 32 filters, a [3*3] grid matrix and ReLU as the activation layer.It is used to capture local patterns in the MFCC sequences.The maxPooling layer is used to downsample the spatial dimensions. Here, the pool matrix is [3*3] pool matrix, strides as (2,2) for the rows and columns and padding the matrix to preserve the spatial information.The layer is then Batchnormalized to improve the training stability and speed.

The output of the final layer is then flattened and passed through a Dense layer with 64 neurons and ReLU as the activation function. The final model is subjected to the Dropout method with dropout probability as 0.3 to prevent overfitting and including randomness into the model.

3.5.4 Compile the network

The model is compiled with an Adam optimizer, sparse categorical crossentropy loss, and accuracy as the evaluation metric. The learning rate for the Adam optimiser is taken as 0.0001.
A summary of the model architecture is printed, providing insights into the layers, parameters, and connections.

3.5.5 Train the network

The model is trained using the training set, with validation performed on the validation set using the 'fit' method of the tensorflow's keras.
Batching, to calculate the gradient of the dataset, is done on a batch size of 32.
The number of epochs is 30,i.e, the model is trained iteratively on the training data 30 times.

Chapter 4

Results Analysis and Testing

Result analysis and testing are critical components of the machine learning model development process. They involve evaluating the performance of a trained model, understanding its strengths and weaknesses, and making informed decisions about its deployment.

4.1 Keras's fit method

In Keras, the fit method is a central component for training a machine learning model.It returns a History object, encapsulating crucial information about the training process. This includes training and validation loss, as well as any additional metrics specified during model compilation. The History object is commonly used for analyzing and visualizing the performance of the model over epochs.
For our MLP classifier, the fit method uses batch size as 32 and the number of epochs as 50.

4.2 Model summary

The model.summary() function was used to generate a summary of the neural network architecture. It provides a quick overview of the layers, their output shapes, and the total number of trainable parameters in the model. This information is instrumental in understanding the complexity and structure of the neural network.

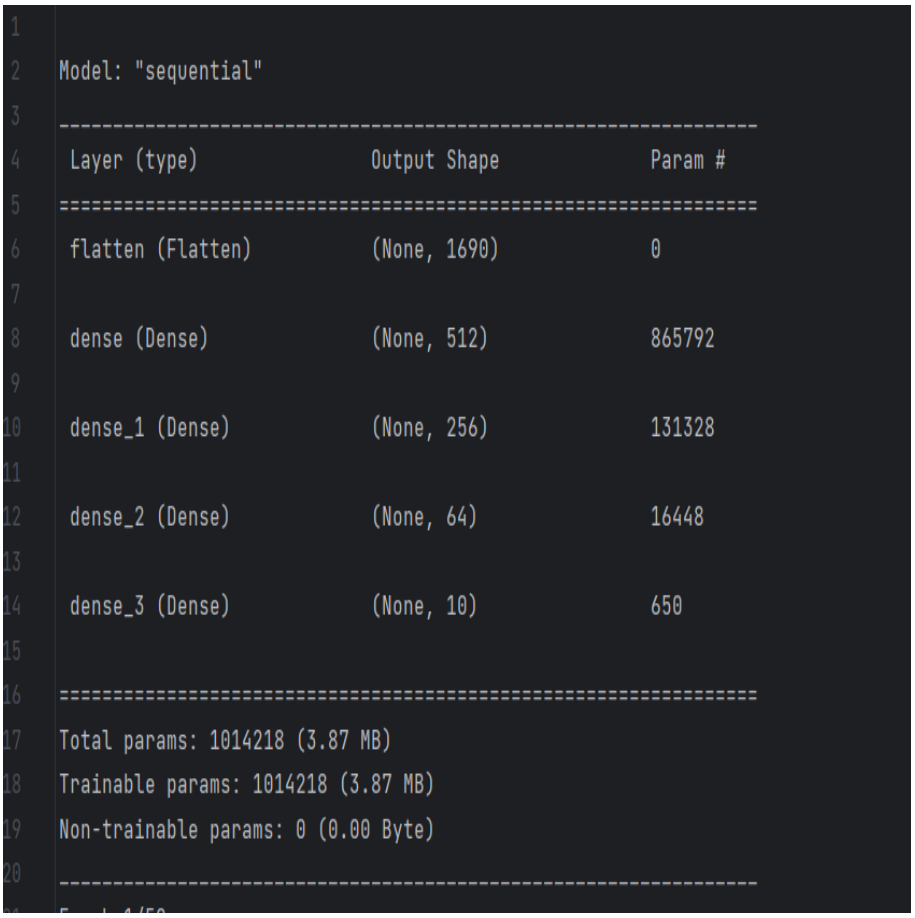


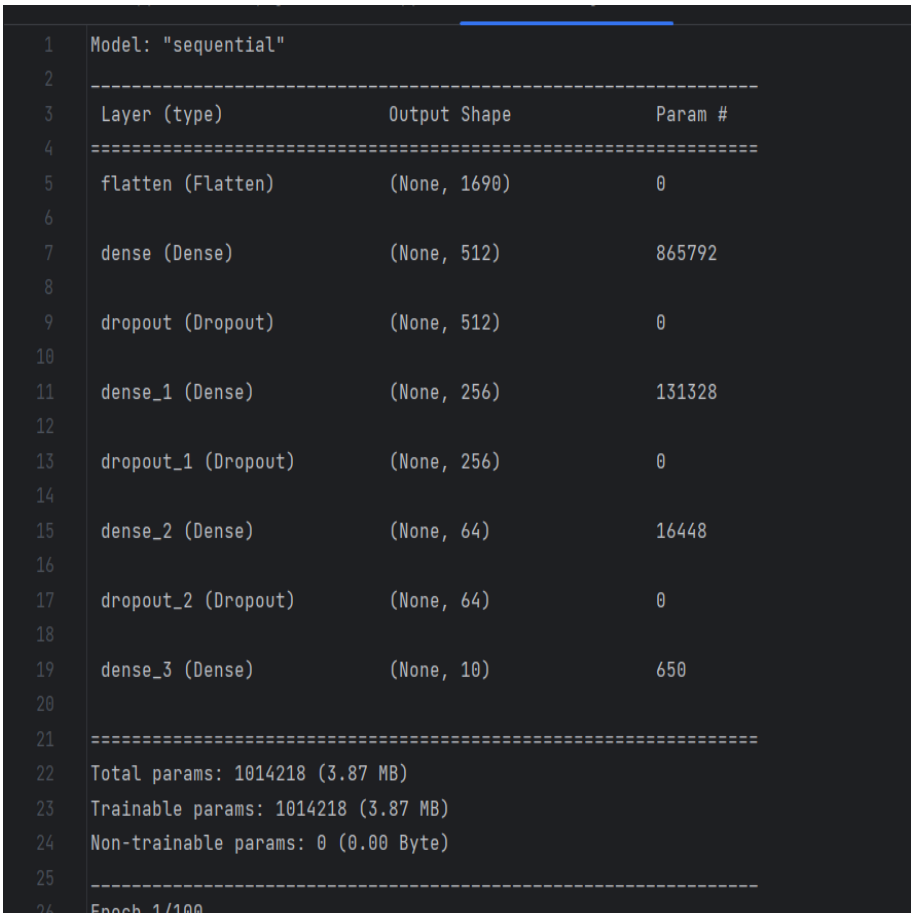
FIGURE 4.1: Model summary of MLP, source: Project code

4.3 The plot_history function

For investigating the training and validation performance of the model, a custom plotting function, plot_history is employed. This function utilizes the training history returned by the fit method, providing insights into the accuracy and error trends over the course of training and validation epochs.

This function plots two graphs with the help of pyplot of the matplotlib library.The first graph plots the 'accuracy' of the training and the test data.The second plot plots the error plot of training and test datas.

Visualizing this data aids in assessing the model's learning dynamics, identifying potential overfitting or underfitting, and guiding further analysis and decision-making.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 1690)              0

 dense (Dense)               (None, 512)               865792

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 64)                16448

 dropout_2 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 1014218 (3.87 MB)
Trainable params: 1014218 (3.87 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/100
```

FIGURE 4.2: Model summary of MLP after solving overfitting, source: Project code

:

```
'''Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 11, 32)       320

 max_pooling2d (MaxPooling2  (None, 64, 6, 32)         0
 D)

 batch_normalization (Batch  (None, 64, 6, 32)         128
 Normalization)

 conv2d_1 (Conv2D)           (None, 62, 4, 32)         9248

 max_pooling2d_1 (MaxPoolin  (None, 31, 2, 32)         0
 g2D)

 batch_normalization_1 (Bat  (None, 31, 2, 32)         128
 chNormalization)

 conv2d_2 (Conv2D)           (None, 30, 1, 32)         4128

 max_pooling2d_2 (MaxPoolin  (None, 15, 1, 32)         0
 g2D)

 batch_normalization_2 (Bat  (None, 15, 1, 32)         128
 chNormalization)

 flatten (Flatten)           (None, 480)               0

 dense (Dense)               (None, 64)                30784
```

FIGURE 4.3: Model summary for CNN model, source: Project code

4.4 Evaluation of model on testset

After the training phase, the model was evaluated on a separate test set using the evaluate method. The test set performance was assessed by computing the test loss and accuracy. The obtained results provide a measure of the model's generalization to previously unseen data, forming a crucial aspect of model assessment and validation. The final test accuracy was reported as [test_acc].

```
1 usage
def plot_history(history):

    fig, axs = plt.subplots(2)

    # create accuracy sublpot
    axs[0].plot(history.history["accuracy"], label="train accuracy")
    axs[0].plot(history.history["val_accuracy"], label="test accuracy")
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy eval")

    # create error sublpot
    axs[1].plot(history.history["loss"], label="train error")
    axs[1].plot(history.history["val_loss"], label="test error")
    axs[1].set_ylabel("Error")
    axs[1].set_xlabel("Epoch")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Error eval")

    plt.show()
```

FIGURE 4.4: The plot_history code, source: Project code

The test accuracy for Multilayer Neural network is 0.594125509262085. The test accuracy for our Convolution Neural Network is 0.7140568494796753.

4.5 The predict function

In order to assess the model's predictions on an individual level, a sample was randomly selected from the test set for prediction. The chosen sample (X_to_predict) and its corresponding true label (y_to_predict) were used as input to the custom predict function. This function employs the trained model to predict the label for the given input and outputs the true and predicted labels for analysis.

The predict function takes the model, input value and the intended output value as the inputs, add one more dimension to the input data, as tensorflow needs a 3-dimensional data and predict a values for the given input value.For this, the model uses the model.predict method on the input value.It returns the probability distribution of output values for the different labels.The final predicted value is the label with the highest probability.The function prints the intended and actual label for the given input.
Such individual predictions offer a granular understanding of the model's behavior

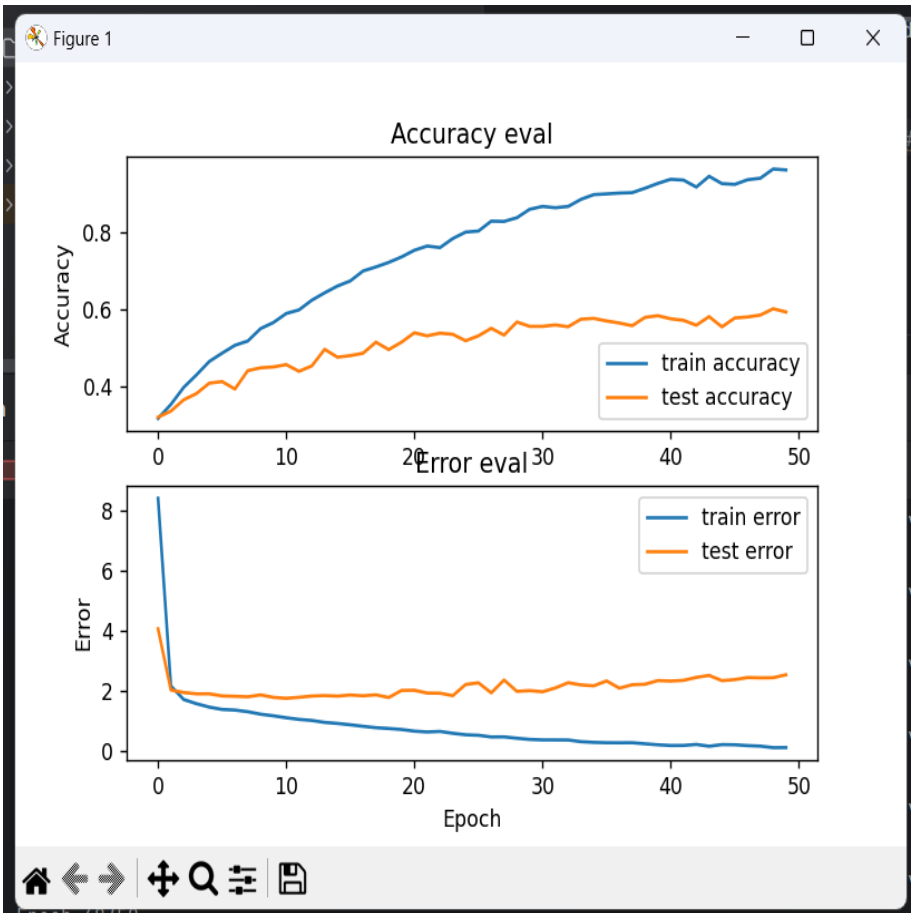


FIGURE 4.5: The MLP result plot of accuracy and error, source: Project code

on specific instances within the test set.

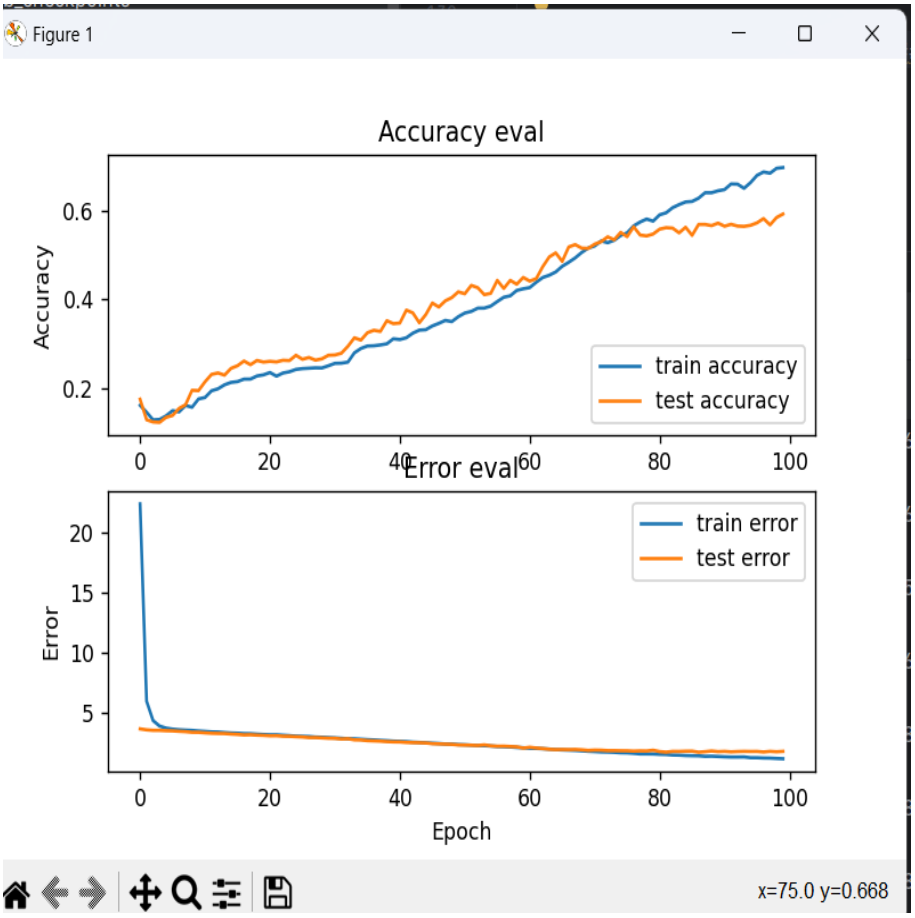


FIGURE 4.6: MLP result plot of accuracy and error after solving overfitting, source: Project code

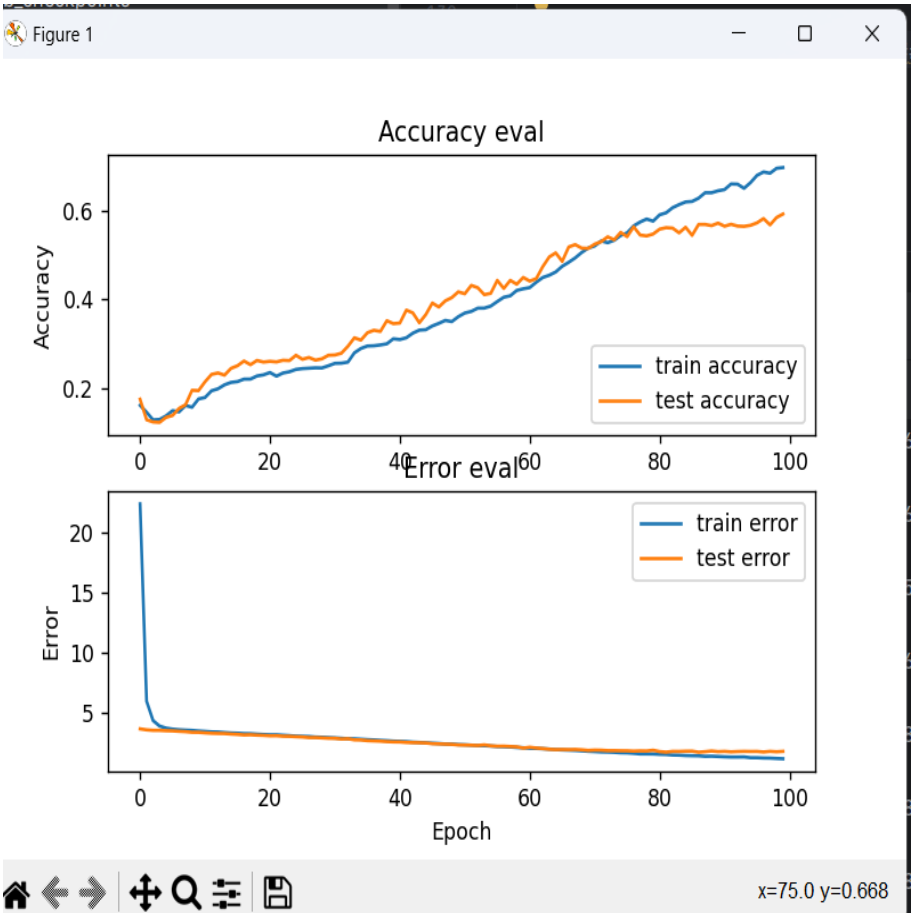
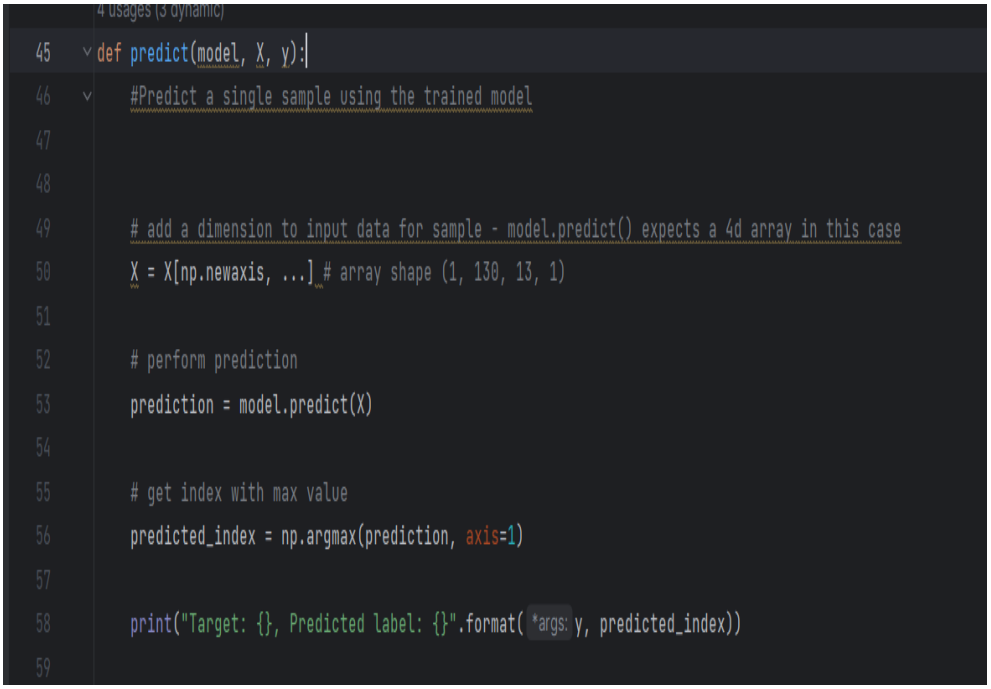


FIGURE 4.7: CNN result plot of accuracy and error, source: Project code



```
4 usages (3 dynamic)
45  def predict(model, X, y):
46      #Predict a single sample using the trained model
47
48
49      # add a dimension to input data for sample - model.predict() expects a 4d array in this case
50      X = X[np.newaxis, ...] # array shape (1, 130, 13, 1)
51
52      # perform prediction
53      prediction = model.predict(X)
54
55      # get index with max value
56      predicted_index = np.argmax(prediction, axis=1)
57
58      print("Target: {}, Predicted label: {}".format( *args: y, predicted_index))
59
```

FIGURE 4.8: The predict function, source: Project code

Chapter 5

Conclusion and Future Works

5.1 Conclusion

In conclusion, the comprehensive implementation and training of two distinct models, namely a Multi-Layer Perceptron (MLP) and a Convolutional Neural Network (CNN), have yielded valuable insights into their respective performances within the realm of music genre classification. Both models underwent rigorous evaluation using the well-established GTZAN dataset, showcasing commendable accuracy in the recognition of diverse music genres.

The MLP model, characterized by its fully connected architecture, demonstrated notable accuracy in effectively classifying music genres. Its ability to discern patterns in the input features, coupled with the universal approximation theorem, underscores the strength of MLPs in capturing intricate relationships within music data.

On the other hand, the CNN model, specifically designed to capture spatial hierarchies in audio data using Mel-frequency cepstral coefficients (MFCCs), exhibited enhanced performance in recognizing complex patterns inherent in music genres. The utilization of MFCCs as input features facilitated the extraction of relevant spectral information, allowing the CNN to discern subtle nuances and hierarchical structures within the audio signals.

However, it is imperative to acknowledge certain limitations inherent in the GTZAN dataset, which may impact the generalization and broader applicability of the models. The dataset's relatively modest size, limited diversity of genres, and short audio clip durations pose challenges in fully capturing the richness and variability present in real-world music databases.

5.2 Future Work

For future work, researchers should consider addressing these limitations and exploring avenues to enhance model robustness and versatility:

- **Dataset Explansion:** Expanding the dataset to encompass a more diverse range of music genres, especially niche or regional styles, is crucial. A broader representation in the training dataset will contribute to a more comprehensive and representative model.

- **Integration of Additional Features:** Beyond genre classification, incorporating additional metadata such as artist information, album details, or release years can offer richer context for music analysis. This expanded set of features can enhance the model's understanding of the music beyond its genre.

- **Exploration of Contemporary Music Styles:** The dynamic landscape of music continually introduces new genres and production styles. Including datasets that capture emerging genres and contemporary production techniques is essential for ensuring the relevance and applicability of the models to the evolving music scene.

  Assessing the models on larger and more recent datasets is paramount. This step allows researchers to gauge the scalability of the models and their effectiveness in handling a broader spectrum of music data, ensuring their adaptability to evolving musical trends.

- **Robustness to Audio Variability:** Enhancing the models' robustness to variations in audio quality is critical. This includes addressing differences in production quality, background noise, and other audio variables. Improving the models' performance under diverse audio conditions will better simulate real-world scenarios.

By addressing these considerations, future iterations of music classifiers can strive for improved performance across a wider range of music genres and production styles, paving the way for more accurate and versatile music analysis applications.

Chapter 6

Bibiliography

Bibliography

[1] Han, H., Lee, G. (2018). Deep Convolutional Neural Networks for Music Genre Classification. *IEEE Transactions on Multimedia, 20(9), 2387-2396.*

[2] A. D. S. Bappy, Li-Chia Yang, and Zhu Li. Deep Learning for Audio-Based Music Classification. *IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2017.*

[3] Tzanetakis, G., Cook, P. (2002). Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing, 10(5), 293-302.*

[4] Music Genre Classification Using Machine Learning Techniques: A Review *Xiaoou Chen and Runwei Ding.*

[5] Kaggle"`www.kaggle.com/`"

[6] Google"`www.google.com`"

[7] W3school"`www.w3schools.com`"

[8] tutorialspoint"`www.tutorialspoint.php`"

[9] stackoverflow"`http://stackoverflow.com`"

Appendix A

User Manual

- **Download and Extract Dataset:** Download the GTZAN dataset and extract it to a folder on your system.
  - **Install Libraries:** Open your terminal or command prompt and install the required libraries using:

    pip install tensorflow librosa matplotlib numpy scikit-learn
  - **Set Up Python Environment:** Install a Python environment such as PyCharm or Visual Studio Code.
-