

CONTENT-BASED ANALYSIS OF MUSIC SIGNALS CLASSIFICATION USING DEEP LEARNING

*A Project Report submitted in the partial fulfillment of the
Requirements for the award of the degree of*

**BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING**

Submitted by

K. L.V. SAI SUBHASH (Regd.no:21NE1A0572)

Under the Esteemed Guidance of

Dr. R. LALU NAIK M.Tech, Ph.D

Professor



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

TIRUMALA ENGINEERING COLLEGE

(Approved by AICTE & Affiliated to JNTU, KAKINADA, Accredited by NAAC & NBA)

Jonnalagadda, Narasaraopet, Palnadu (Dt), A.P.

2021-2025

TIRUMALA ENGINEERING COLLEGE

(Approved by AICTE & Affiliated to JNTUKAKINADA, Accredited by NAAC & NBA)

Jonnalagadda, Narasaraopet-522601, Palnadu (Dist) A.P

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that the project report entitled “**CONTENT-BASED ANALYSIS OF MUSIC SIGNALS CLASSIFICATION USING DEEP LEARNING**” is the bonafide work done by **K. L. V. SAI SUBHASH (21NE1A0572)** in partial fulfillment of the requirements for the award of “**Bachelor of Technology**” degree in the Department of **COMPUTER SCIENCE AND ENGINEERING** from Jawaharlal Nehru Technological University, Kakinada, during the year 2024 - 2025 under our guidance and supervision and worth of acceptance of requirements of the university

Project Guide

Dr. R. LALU NAIK **M.Tech, Ph.D.**

Head of the Department

Dr.K. SATHISH **M.Tech, Ph.D.**

Project Coordinator

Mr. S. ANIL KUMAR **M.Tech (Ph.D).,**

External Examiner

ACKNOWLEDGEMENT

I wish to express my sincere thanks to various personalities who are responsible for the completion of the project. We extremely thankful to our principal sir **Dr. Y. V. Narayana M.E., Ph. D, FIETE** for his kind attention and valuable guidance throughout the course.

I express my deep felt gratitude to our **H.O.D Dr. K. Sathish M. Tech, PhD.** and **Mr. S. Anil Kumar M. Tech, (Ph.D)**, coordinator of the project for extending their encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout the project work.

I wish to express our sincere deep sense of gratitude to our guide **Dr. R. Lalu Naik M.Tech, Ph.D.** for significant suggestions and help in every respect to accomplish the project work. Her persisting encouragement, everlasting patience and keen interest in discussions have benefited to us.

I extend my sincere thanks to all other teaching and non-teaching staff of department of Computer Science and Engineering for their cooperation and encouragement throughout my Bachelor's degree. I have no words to acknowledge the warm affection, constant inspiration and encouragement that I received from my parents.

I affectionately acknowledge the encouragement received from my friends and those who involved in giving valuable suggestions had clarifying out doubts which had really helped us in successfully completing my project.

BY

K. L.V. SAI SUBHASH

(Regd.no:21NE1A0572)

ABSTRACT

ABSTRACT

Content-based analysis of music signals classification automatically categorizes music into different genres based on various musical attributes and features in a small number of music files. This is a crucial problem in the field of music information retrieval as it provides a way to organize and analyze large amounts of music files. MGC can be performed using conventional machine learning algorithms such as SVM, k-nearest neighbors, Decision trees, and neural networks. These algorithms learn to recognize different musical features and attributes to categorize the music files into different genres.

Content-based analysis of music signals classification automatically categorizes music into different genres based on various musical attributes and features in a small number of music files. This is a crucial problem in the field of music information retrieval as it provides a way to organize and analyze large amounts of music files. MGC can be performed using conventional machine learning algorithms such as SVM, k-nearest neighbours, Decision trees, and neural networks. These algorithms learn to recognize different musical features and attributes to categorize the music files into different genres. The literature shows that the performance of conventional machine learning algorithms is inferior to deep learning algorithms such as CNN, RNN, etc., in various applications. Hence, the CNN algorithm is adapted to implement the classification of music files. This aims to classify music genres using CNN deep learning techniques. The performance of the algorithms for MGC can be evaluated using metrics such as accuracy, precision, recall, and F1-score. Additionally, the impact of different features and algorithms on the performance of MGC can be studied and compared. It has applications in areas such as automated music recommendation systems, music education, and music production. An accuracy of 83% is achieved by using CNN to accomplish the task of MGC.

INDEX

<u>CONTENT</u>	<u>PAGENO</u>
1 Introduction	1-4
1.1 Introduction	1-2
1.2 Scope of the Project	3-4
2 Literature Survey	5-6
3 System Analysis	7-30
3.1 Existing System	7
3.2 Proposed System	8
3.3 System Requirements	9
3.4 Analysis	10-12
3.5 Algorithms	13-24
3.6 Feasibility Study	25-29
3.6.1 Technical Feasibility	26
3.6.2 Operational Feasibility	27
3.6.3 Economic Feasibility	28-29
3.7 Data Preprocessing	30
4 Design	31-34
4.1 Class Diagram	31
4.2 Use Case Diagram	32
4.3 Activity Diagram	33
4.4 Component Diagram	34
5 Implementation	35-44
6 Result Analysis	45
7 Output Screenshots	46-52
8 Conclusion	53
9 Future Enhancement	54
10 Bibliography	55

INTRODUCTION

1.1 Introduction to Project

Music is a form of expression that combines melodies and sounds to generate aesthetic and emotive experiences for listeners. It can be created with various techniques and instruments, such as stringed instruments, wind-based instruments, and digital instruments. Music is classified into genres depending on characteristics such as melody, timbre, tempo, rhythm, as well as harmony. Rock, blues, classical, pop, hip-hop, electronic, jazz, and country music are some well-known musical genres. Each genre has distinct characteristics that are typically associated with particular cultural, historical, and social contexts.

Central to the organization and categorization of music is the concept of genre—a multifaceted construct that encapsulates a spectrum of stylistic conventions, thematic motifs, and sonic signatures. Music genre classification, the cornerstone of music information retrieval (MIR), endeavors to automate the process of categorizing musical compositions into distinct genres based on their inherent attributes and stylistic characteristics. By unraveling the intricate tapestry of musical genres, classification algorithms pave the way for personalized recommendations, curated playlists, and enhanced user experiences in the digital realm.

In this ambitious undertaking, we embark on a voyage of exploration into the realm of music genre classification using deep learning to distill the essence of musical genres from raw audio data. Our arsenal includes Convolutional Neural Networks (CNNs) and Artificial Neural Networks (ANNs), two potent techniques that have revolutionized the landscape of machine learning and computational music analysis.

Unlike traditional rule-based systems that rely on handcrafted features and heuristic algorithms, deep learning models possess the remarkable capacity to autonomously extract intricate patterns and latent representations from unstructured audio signals, thereby transcending the limitations of human-defined feature sets. CNNs, inspired by the hierarchical structure of the visual cortex, excel at capturing spatial and temporal patterns in data, making them well-suited for tasks such as image recognition and audio analysis.

In parallel, ANNs offer a versatile framework for modeling complex relationships and capturing high-level abstractions in data. By constructing multi-layer perceptrons with interconnected neurons, ANNs can learn nonlinear mappings between input features and output labels, enabling them to discern intricate patterns and subtle variations in music signals. Through a synthesis of cutting-edge research, innovative methodologies, and empirical experimentation, we aim to harness the expressive power of CNNs and ANNs to unravel the underlying structures and patterns that define each.

Our journey unfolds against the backdrop of a rich and diverse musical landscape, spanning a multitude of genres, each imbued with its own unique sonic palette and cultural significance. From the soulful strains of Blues to the intricate harmonies of Classical, from the twangy melodies of Country to the pulsating rhythms of Disco, we traverse the vast expanse of musical expression, seeking to unravel the underlying structures and patterns that define each genre.

As we embark on this exhilarating journey, we are confronted with a myriad of challenges and complexities inherent to the domain of music genre classification. The subjectivity of genre boundaries, the diversity of musical styles, and the inherent ambiguity of artistic expression pose formidable obstacles on our quest for computational proficiency.

Throughout this project, we adopt a multidisciplinary approach, drawing insights from music theory, cognitive science, signal processing, and machine learning to enrich our understanding and refine our methodologies. From the intricacies of feature engineering to the nuances of model selection and evaluation. By the culmination of our odyssey, we aspire not only to advance the frontiers of computational music analysis but also to deepen our appreciation for the boundless creativity and cultural richness embodied in the myriad genres of music. Whether you are a seasoned researcher, a budding enthusiast, or an intrepid explorer in the realm of artificial intelligence, we invite you to join us on this transformative journey as we unravel the symphony of genres that defines the tapestry of human musical expression. Let the harmonies of deep learning guide us as we embark on this exhilarating voyage into the realm of music genre classification.

1.2 Scope of the project

The main aim is to create a Deep learning model, which classifies music samples into different genres. It aims to predict the genre using an audio file as its input

- 1. Data Collection:** Gathering a dataset of music videos or audiovisual content covering various genres. This dataset should be diverse and representative of different music genres.
- 2. Feature Extraction - Audio:** As in traditional music genre classification, extract audio features such as spectrogram analysis, MFCCs, rhythm analysis, etc., from the audio track of each music video.
- 3. Feature Extraction - Visual:** Extract visual features from the video content. This could include frame-level analysis, color histograms, texture features, motion analysis, deep learning-based features extracted from pre-trained convolutional neural networks (CNNs) like VGG, ResNet, or custom architectures.
- 4. Integration of Audio-Visual Features:** Combine the audio and visual features into a unified feature representation for each music video. This can be done through concatenation, fusion, or other integration techniques.
- 5. Labeling:** Assign genre labels to each music video in the dataset. This may require manual labeling or utilizing existing metadata.
- 6. Model Building:** Develop machine learning or deep learning models to learn patterns from the integrated audio-visual features and predict the genre of a given music video. This could involve architectures like multimodal neural networks, which are designed to handle both audio and visual inputs simultaneously.
- 7. Training and Validation:** Split the dataset into training and validation sets, train the model, and evaluate its performance using appropriate evaluation metrics for multi-modal classification tasks.
- 8. Evaluation Metrics:** Assess the performance of the model using metrics such as accuracy, precision, recall, F1-score, or area under the ROC curve (AUC).
- 9. Hyperparameter Tuning:** Optimize the model hyperparameters to improve performance, considering both audio and visual aspects.

10. Testing: Evaluate the model on a separate test dataset to assess its generalization performance and ensure it can classify unseen music videos accurately.

11. Deployment: Integrate the trained model into an application or system where users can upload music videos and get their genres classified automatically.

12. Monitoring and Maintenance: Continuously monitor the model's performance and update it as necessary to adapt to changes in music trends or user preferences, considering both audio and visual aspects.

Incorporating visual analysis alongside audio analysis can provide richer information for genre classification, potentially leading to more accurate and robust models.

LITERATURE SURVEY

2. LITERATURE SURVEY

1. "Music Genre Classification: A Taxonomy of Computational Approaches" by G. Tzanetakis and P. Cook:

- This paper provides a comprehensive overview of various computational approaches to music genre classification, including feature extraction, feature selection, and classification techniques. It covers both traditional methods and more recent developments in the field.

2. "A Survey of Music Information Retrieval Systems" by J. Stephen Downie:

- While not solely focused on genre classification, this survey paper provides a broad overview of music information retrieval (MIR) systems, which often include genre classification as a component. It covers a wide range of topics related to MIR, including audio feature extraction, similarity measures, and evaluation metrics.

3. "Content-Based Music Genre Classification: A Survey" by Wei Li et al.:

- This survey paper focuses specifically on content-based music genre classification and provides an in-depth analysis of various feature extraction techniques, classification algorithms, and evaluation methods used in the field. It also discusses challenges and future directions for research in music genre classification.

4. "Deep Learning Techniques for Music Genre Classification: A Survey" by H. S. Gupta et al.:

- As deep learning has become increasingly popular in music genre classification, this survey paper provides an overview of deep learning techniques applied to this task. It covers convolutional neural networks (CNNs), recurrent neural networks (RNNs), and their variants, as well as recent advancements and challenges in the field.

5. A Review of Audio-Based Music Classification and Annotation by M. S. Kam et al.:

- While not exclusively focused on genre classification, this review paper discusses various audio-based music classification and annotation tasks, including genre classification, mood detection, and artist identification. It provides insights into feature extraction, classification algorithms, and evaluation methodologies used in these tasks. These papers should give you a solid foundation for understanding the state of the art in music genre classification and the different approaches researchers have taken to tackle this problem.

6. "Music Genre Classification: A Review" by M. S. Kam et al. (2016):

- This paper provides a comprehensive review of the state-of-the-art techniques and methodologies in music genre classification. It covers various aspects such as feature extraction, classification algorithms, evaluation metrics, and datasets commonly used in the field.

7. Music genre recognition with deep neural networks Albert Jimenez, Ferran Jos é in 2018 published. Music genre recognition with deep neural networks [1]. The methods they used are deep learning and convolutional neural networks, multiframe and transfer learning. The main advantage of this is the multiframe approach. Audio files are placed in a confusion matrix to find the mean of all frames. So the classification of genre is more accurate. They have used a two layer convolutional network with mel-spectrogram technique and also raw audio signals as their input features. For training and fine-tuning the dataset optimizers like adaptive learning rate ADAM and SGD are used. Comparing the results ADAM is chosen to use. On the other hand, when it uses a large amount of data, it takes more time for processing, which is a drawback of this paper

8. Survey on Transfer Learning Sinno Jialin Pan and Qiang Yang Fellow in 2010 published —Survey on Transfer Learning[2]. They have used transfer learning, machine learning and data mining techniques. Transfer learning is a technique in which people learn to apply the knowledge they have gained so far in every problem they face in their day to day life. The transfer learning technique is closely related to multi- task learning methods.

SYSTEM ANALYSIS

3. SYSTEM ANALYSIS

3.1 Existing System

Several studies on music genre classification using CNN and SVM have been conducted. Choi et al. (2016) used CNN for music genre classification using a dataset of 2000 audio files in their study. The proposed model had a 65% accuracy. Han et al. (2018) used a CNN- SVM hybrid approach for music genre classification using the GTZAN Dataset in another study. The proposed model's accuracy was 85.9%. Several studies have been conducted in recent years to investigate the use of various machine learning algorithms and techniques to improve accuracy in music genre classification. Convolutional Neural Networks (CNN) and Support Vector Machines (SVM) are two popular techniques (SVM)

DISADVANTAGES

- Data requirements : Difficult to meet all the requirements of data
- Computational resources : Needs huge amount of computational resources
- Interpretability : Complex data maylead to misinterpretations.
- Overfitting : Leads to poor generalization and inaccurate predictions on unseen data.
- Domain-specific challenges : Complicates the adaptation of generic models to specific industries.

3.2 PROPOSED SYSTEM

Music has an important role in everyone's life. People have their preferences in music. It is highly variable and subjective in nature. It is important to classify music according to their genre this can be achieved using automated music genre classification using deep learning techniques. There are potential overlaps between genres which make it difficult to identify them. Manual genre classification is a hectic task. Therefore, developing an accurate and robust music genre classification model requires thorough domain knowledge. Hence, automated process of MGC is done using CNN. Music has an important role in everyone's life. People have their preferences in music.

It is highly variable and subjective in nature. It is important to classify music according to their genre this can be achieved using automated music genre classification using deep learning techniques. There are potential overlaps between genres which make it difficult to identify them. Manual genre classification is a hectic task. Therefore, developing an accurate and robust music genre classification model requires thorough domain knowledge. Hence, automated process of MGC is done using CNN.

ADVANTAGES

- Automated classification
- Scalability
- Feature learning
- Adaptability
- High accuracy

3.3 SYSTEM REQUIREMENTS

1.3.1 Hardware Requirements:

- Processor : Intel(R) Core™2 i7-5500U CPU @2.50GHz
- RAM : 8GB(gigabyte)
- System Type : 64-bit operating system, x64-based processor

1.3.2 Software Requirements:

- Browser : Any Latest browser like Chrome
- Operating System : Windows 10
- Language : Python
- Platform : Google COLAB

3.4 ANALYSIS

The GTZAN Dataset - Music Genre Classification is a popular dataset for research on music genre classification. It consists of 1000 audio files, each 30 seconds long, from ten different music genres: blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock. The dataset is freely available on Kaggle and is widely used in research to test music genre classification algorithms. The GTZAN dataset contains audio files in WAV format with a sample rate of 22050 Hz and a bit depth of 16 bits. The audio files were sampled from the Million Song Dataset and preprocessed to ensure high quality and the absence of irrelevant noise. The GTZAN Dataset - Music Genre Classification is a balanced dataset, with each music genre containing 100 audio files. This makes the dataset suitable for evaluating the performance of music genre classification algorithms. The dataset has been widely used in research and has become a benchmark dataset for evaluating the performance of different machine learning algorithms.

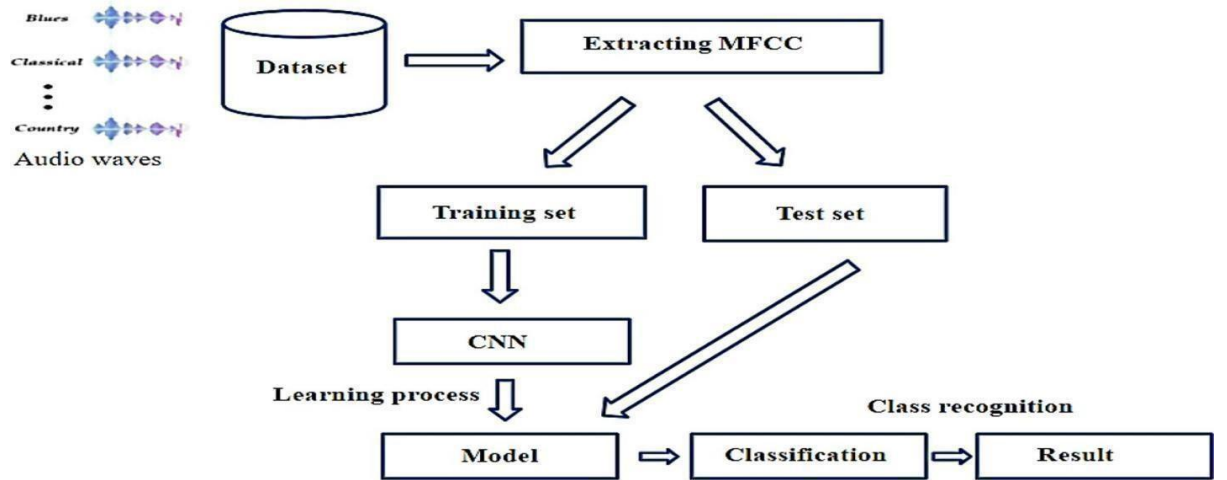


Fig. 1. Architecture diagram

Figure shows the architecture diagram of the proposed work. The dataset contains audio files from 10 different genres. MFCCs (Mel-Frequency Cepstral Coefficients) are extracted from the audio files which will be saved in a JSON file.

These MFCCs are divided into training sets and test sets. The training set is used to train the CNN model. The trained model is then used for the classification of new audio files. The test set is given to the trained model for evaluation

Modularization

The proposed method consists of 4 modules. They are audio visualization, data processing, classification, and evaluation. In this section, we discuss the inputs provided to each module, the processing that happens, and finally, the output of the module.

Audio visualization

Audio visualization is the display of audio data or its representation in a visual form. It is a method of converting audio signals into visual elements.

- Visualizing audio file as a waveform: Plotting an audio in the waveform includes representing the amplitude of the audio as a function of time. Figure 2 shows the waveform.

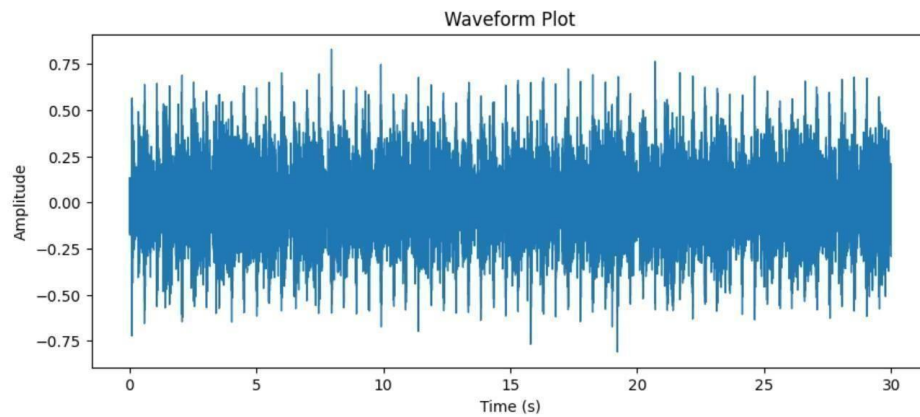


Fig. 2. Waveform Visualization.

Visualizing audio file as a spectrogram: A spectrogram is a visual representation of the frequency spectrum of a signal over time. It is a two-dimensional graph with time on x-axis and frequency on y-axis. The plot is shown in Figure 3.

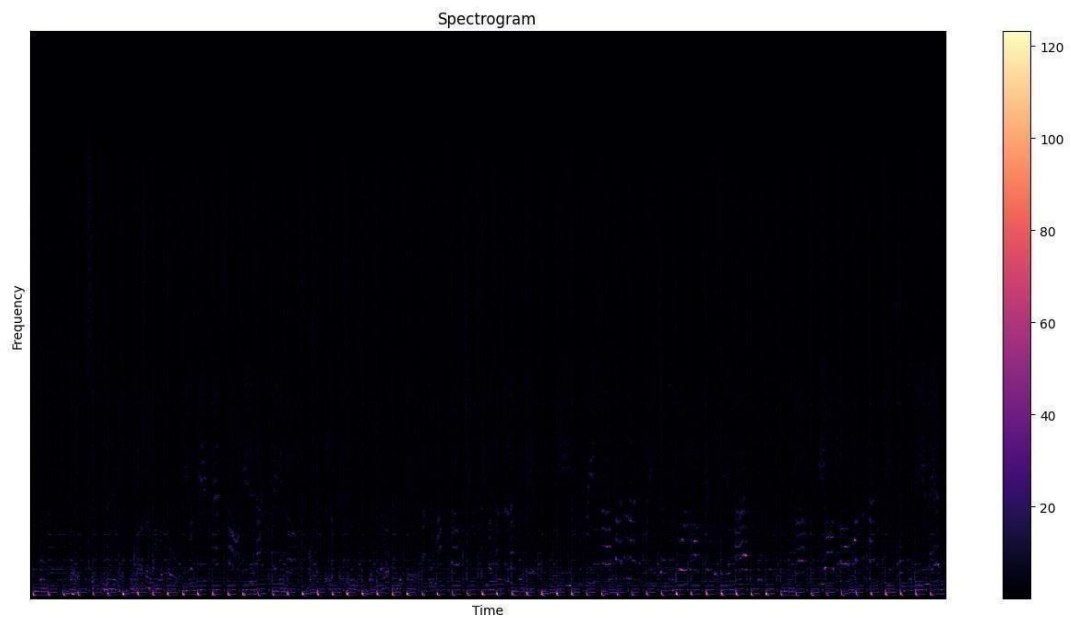
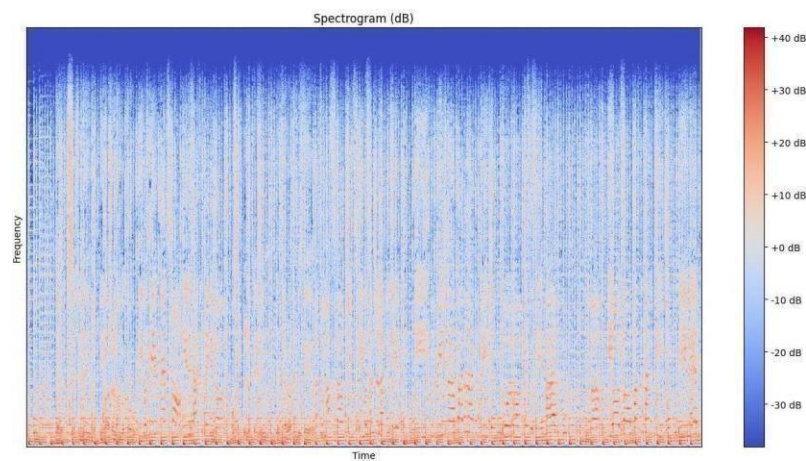


Fig. 3. Spectrogram visualization.

- Visualizing audio file as a Mel-spectrogram: Traditional spectrograms use a linear frequency scale. We can visualize the Mel spectrogram plot through matplotlib. The plot is shown in Figure 4.

Fig. 4. Mel-spectrogram visualization.



3.5 ALGORITHMS

Deep Learning

Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

Deep Learning architectures such as deep neural networks, deep belief networks, graph neural networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance.

Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems. ANNs have various differences from biological brains. Specifically, neural networks tend to be static and symbolic, while the biological brain of most living organisms is dynamic (plastic) and analogue.

The adjective "**deep**" in deep learning refers to the **use of multiple layers in the network**. Early work showed that a linear perceptron cannot be a universal classifier, but that a network with a non-polynomial activation function with one hidden layer of unbounded width can. Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions. In deep learning the layers are also permitted to be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the "structured" part.

Some Deep Learning Types

1. Simple Artificial Neural Network (ANN)

ANN is not directly under the human control computing system which is planned to energize in the same way as the human brain works to support and process information. Processing unit built Artificial neural network composed of input and output. The input units get numerous forms and various structural information are built on an internal weight system and the neural network seeks to acquire knowledge about the content given to construct the output. Firstly, the Artificial Neural Network undergoes the phase of training which learns to understand the patterns of the dataset. Information that passes through the network transforms the structure of the ANN.

2. Convolution Neural Network

- CNN is one of the variations of the multilayer perceptron.
- CNN can contain more than 1 convolution layer and since it contains a convolution layer the network is very deep with fewer parameters.
- CNN is very effective for image recognition and identifying different image patterns.

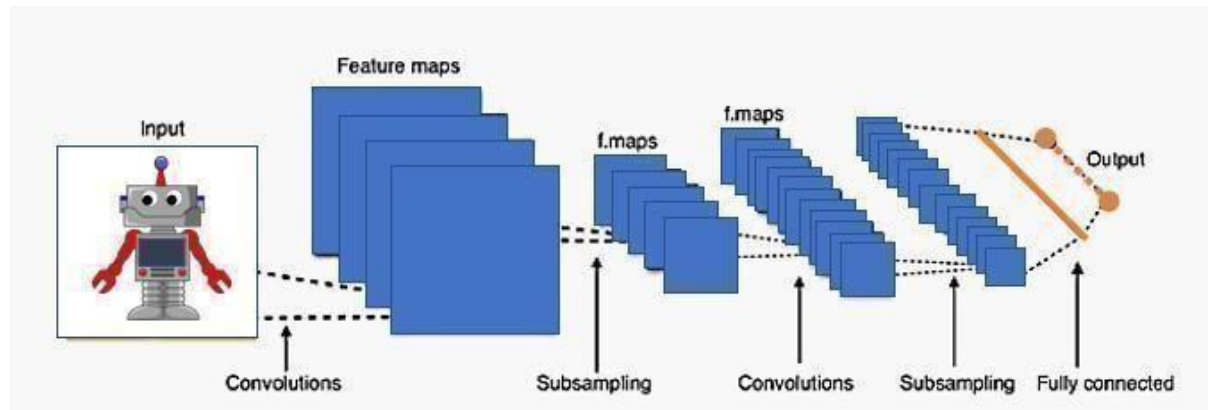


Fig: Convolution Neural Network

3. Recurrent Neural Network:

RNN is a type of neural network where the output of a particular neuron is fed back as an input to the same node.

- These method helps the network to predict the output.
- This kind of network is useful in maintaining a small state of memory which is very useful for developing the chatbot
- This kind of network is used in chatbot development and text to speech technologies

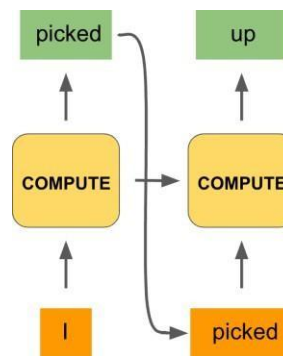


Fig: Recurrent Neural Network

3.5 Applications of Deep Learning:

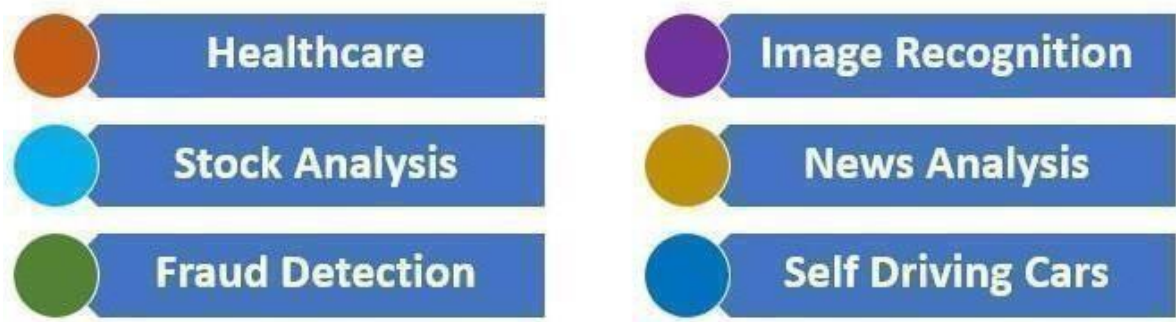


Fig: Applications of Deep Learning

Characteristics of Deep Learning

1. Supervised, Semi-Supervised or Unsupervised

When the category labels are present while you train the data then it is Supervised learning. Algorithms like Linear regression, Logistic regression, decision trees use Supervised Learning. When category labels are not known while you train data then it is unsupervised learning.

2. Huge Amount of Resources

It needs advanced Graphical Processing Units for processing heavy workloads. A huge amount of data needs to be processed like Big data in the form of structured or unstructured data. Sometimes more time also required to process the data, it depends on the amount of data fed in.

3. Large Amount of Layers in Model

A huge amount of layers like input, activation, the output will be required, sometimes the output of one layer can be input to another layer by making few small findings and then these findings are summed up finally in the softmax layer to find out a broader classification for final output.

4. Optimizing Hyper-parameters

Hyperparameters like no of epochs, Batch size, No of layers, Learning rate, needs to be tuned well for successful Model accuracy because it creates a link between layer predictions to final output prediction. Over-fitting and under-fitting can be well handled with hyper- parameters.

5. Cost Function

It says how well the model performance in prediction and accuracy. For each iteration in Deep Learning Model, the goal is to minimize the cost when compared to previous iterations. Mean absolute error, Mean Squared Error, Hinge loss, Cross entropy are different types according to different algorithms used.

6. Optimizing Hyper-parameters

Hyperparameters like no of epochs, Batch size, No of layers, Learning rate, needs to be tuned well for successful Model accuracy because it creates a link between layer predictions to final output prediction. Over-fitting and under-fitting can be well handled with hyper- parameters.

7. Cost Function

It says how well the model performance in prediction and accuracy. For each iteration in Deep Learning Model, the goal is to minimize the cost when compared to previous iterations. Mean absolute error, Mean Squared Error, Hinge loss, Cross entropy are different types according to different algorithms used.

8. Optimizing Hyper-parameters

Hyperparameters like no of epochs, Batch size, No of layers, Learning rate, needs to be tuned well for successful Model accuracy because it creates a link between layer predictions to final output prediction. Over-fitting and under-fitting can be well handled with hyper- parameters.

9. Cost Function

It says how well the model performance in prediction and accuracy. For each iteration in Deep Learning Model, the goal is to minimize the cost when compared to previous iterations. Mean absolute error, Mean Squared Error, Hinge loss, Cross entropy are different types according to different algorithms used.

Advantages of Deep Learning

- Solve Complex problems like Audio processing in Amazon echo, Image recognition, etc, reduce the need for feature extraction, automated tasks wherein predictions can be done in less time using Keras and Tensorflow.
- Parallel computing can be done thus reducing overheads.
- Models can be trained on a huge amount of data and the model gets better with more data.
- High-Quality Predictions when compared with humans by training tirelessly.
- Works well-unstructured data like video clips, documents, sensor data, webcam data, etc.

Deep Learning Algorithms

To build this architecture following algorithms are used:

1. Back Propagation

In this algorithm, we calculate partial derivatives. In general, the gradient descent method for optimization, derivatives (gradients) are calculated at each iteration. In deep learning, functions are not simple; they are the composition of different functions. In this case, it is hard to calculate gradients, so we use approximate differentiation to calculate derivatives. The more the number of parameters, the more expensive approximate differentiation will be.

2. Stochastic Gradient Descent

In Gradient descent, the goal is to find global minima or optimum solution. But to get that, we have to consider local minima solutions (not desirable) also. If the objective function is a convex function, it is easy to find the global minima. The initial value for the function and learning rate are deciding parameters for finding global minima. This can easily be understood by considering a river from the mountain top and searching for a foothill (global minima). But in the way, there will be some ups and downs (local minima) which must be avoided. The river originating point and speed (initial value and learning rate in our case) are deciding factors to find global minima.

3. Learning Rate

The learning rate is like the speed of the river; it can reduce training time and increase performance. In general, to learn any technique/sport, in the beginning, the learning rate is relatively high than at the end when one is to master it. After the intermediate stage, the learning will be slow. The same is applied in deep learning; too large changes are tackled by a higher learning rate and by slowly decreasing the learning rate later for fine-tuning.

4. Batch Normalization

In deep learning initial value of weight (randomly chosen) and learning rate is defined for a minibatch. In the beginning, there would be many outliers, and during backpropagation, these outliers must be compensated to compute the weights to get output. This compensation results in extra epochs. So to avoid it, we use batch normalization.

5. Drop Out

In deep learning, we generally encounter the problem of overfitting. Overfitting in large networks with several parameters makes it difficult to predict on test data. So, to avoid that, we use the dropout method, which drops random units during training by creating different thinned networks'. When testing these thinned networks' predictions are averaged, which helps to avoid overfitting.

6. Bag of Words

We use a continuous bag of words to predict the next word. For e.g., we see in email writing the autosuggestion for completing the sentence is part of NLP. This is done by considering lots of sentences and for a specific word surrounding words that are captured. These specific words and surrounding words are fed to the neural network. After the training model, it can predict the specific word based on the surrounding words.

7. Long Short Term Memory

LSTM is very useful in sequence prediction problems like language translation, predicting sales and finding the stock price. LSTM has the edge over other techniques because it is able to consider previous data. LSTM makes modification by cell states mechanism. It remembers to forget things. The 3 main aspects of LSTM make it stand out from other deep learning techniques. First is when the neuron should have input, second when to remember previous data and what to forget and third is when to pass output.

Libraries of Deep Learning

All the libraries which are generally used for deep learning are open source and few of them are as follows:

- TensorFlow
- deeplearning4j
- Torch
- Caffe
- Microsoft CNTK
- ML.NET

1. TensorFlow

- Different types of deep nets can be developed and also the various packages available in this library are used to attain and address most of the tasks and problems in the field of deep learning.
- Due to a flexible computational graphical structure of TensorFlow, this library is not only limited to deep learning operations it can be used for many different operations and applications.
- TensorFlow provides a layer or we can say more of a wrapper, known as Keras which is used to access the different packages and methods easily of TensorFlow. .
- Google has developed this library for Mobil platforms as well which is known as TensorFlow lite.

2. Deeplearning4j

- Deeplearning4j is the open-source java library which only supports java programming language and this library is written in Java.
- This was developed by Adam Gibson to provide distributed multimode capabilities for deep neural networks.
- This library is very much use full for the application which is having build on top of big data.
- This library works with Scala and also provide inbuilt GPU support.

3. Torch

- This open-source deep-learning library was developed by Facebook and Twitter.
- This library is written in Lua programming language.
- However PyTorch is the library which is widely used, and it's written in a python programming language.

4. Caffe

- Caffe is an open-source deep-learning library written in C++/CUDA and developed by Yangqing Jia of Google.
- This library was first developed for computer vision tax.
- Caffe gives permission to the user to configure the hyperparameters for a deep net.
- The layer configuration is very robust and very much sophisticated.

5. Theano

- This is the open-source deep-learning library written in Python and CUDA.
- This library is very similar to the TensorFlow library but the implementation and usage are not that simple as that of TensorFlow.
- Theano is not that easy to use and many deep learning libraries extend the features of this library to help ease the life of the developer for coding the deep learning models.
- Theano is fastest amongst most of the libraries mentioned because it makes use of vectors and matrices for all the functions and the vectorized code runs faster since parallel processing for the multiple values makes the things faster.

6. Microsoft CNTK

- This is a cognitive toolkit developed by Microsoft to venture in the field of Artificial intelligence.
 - This library is written in python and its supports the other packages and libraries which python programming language supports, and it comes with Microsoft visual studio.
- CNTK is used to describe neural networks as a series of computational directed graphs.

7. ML.NET

- ML.NET is the open-source library which is also developed by Microsoft for the dot net developers.
- This library is written in C# and F# and it uses the Microsoft dot net platform.
- With the library, it becomes easy to create desktop as well as large scale web applications which can bring the vast possibility of the machine learning algorithm to the end-user.

Layers in Convolutional Neural Networks

Below are the Layers of convolutional neural networks:

1. Image Input Layer:

The input layer gives inputs (mostly images), and normalization is carried out. Input size has to be mentioned here.

2. Convolutional Layer:

Convolution is performed in this layer. The image is divided into perceptrons (algorithm); local fields are created, leading to the compression of perceptrons to feature maps as a matrix with size $m \times n$.

3. Non-Linearity Layer:

Here feature maps are taken as input, and activation maps are given as output with the help of the activation function. The activation function is generally implemented as sigmoid or hyperbolic tangent functions.

4. Rectification Layer:

The crucial component of CNN, this layer does the training faster without reducing accuracy. It performs element-wise absolute value operation on activation maps.

5. Rectified Linear Units (ReLU):

ReLU combines non-linear and rectification layers on CNN. This does the threshold operation where negative values are converted to zero. However, ReLU doesn't change the size of the input.

6. Pooling Layer:

The pooling layer is also called the downsampling layer, as this is responsible for reducing the size of activation maps. A filter and stride of the same length are applied to the input volume. This layer ignores less significant data; hence image recognition is done in a smaller representation. This layer reduces overfitting. Since the amount of parameters is reduced using the pooling layer, the cost is also reduced. The input is divided into rectangular pooling regions, and either maximum or average is calculated, which returns maximum or average consequently. Max Pooling is a popular one.

7. Dropout Layer:

This layer randomly sets the input layer to zero with a given probability. More results in different elements are dropped after this operation. This layer also helps to reduce overfitting. It makes the network to be redundant. No learning happens in this layer. This operation is carried out only during training.

8. Fully Connected Layer:

Activation maps, which are the output of previous layers, is turned into a class probability distribution in this layer. FC layer multiplies the input by a weight matrix and adds the bias vector.

9. Output Layer:

FC layer is followed by softmax and classification layers. The softmax function is applied to the input. The classification layer computes the cross-entropy and loss function for classification problems.

10. Regression Layer:

Half the mean squared error is computed in this layer. This layer should follow the FC layer.

Common steps for any Tensorflow based Algorithms:

The basic steps of TensorFlow algorithm are:

Step 1: Data is Imported/Generated: TensorFlow Models depends heavily on the huge amount of Data. Either you can import your own dataset or TensorFlow also comes with the collection of Type this command to check out available datasets in TensorFlow.

```
import TensorFlow as tf
```

```
import TensorFlow_datasets as tendata
```

```
#This command will generate a list of datasets available in the TensorFlow
```

```
print(tfds.list_builders())
```

Step 2: Data Normalization or Transformation: If the data is not in the appropriate forum. The Batch Normalization is the command approach used to normalize data in the Tensor Flow

Step 3: Set the Parameters of the Algorithm: For eg; the number of Iterations, Learning rate, etc.

Step 4: Set and initialize the variables and Placeholders: Variables and Placeholders are two basic programming Elements of the Tensor Flow. Variables hold the state of the graph and placeholders are used to feed the data in the graph at the later date.

Step 5: Create Model structure: What operations will be performed on the data is defined.

Step 6: Define the Loss Function: It calculates the difference between predicted values and actual values. It tells how well your model is trained basically used to evaluate the output.

Step 7: Train Model: Initialize computational graph and create an Instance of a graph. Feed data into the model with the help of placeholders and let the Tensor Flow do the rest of the processing for better predictions.

Step 8: Evaluate the performance: Evaluate the model by checking with new data.

Step 9: Predict the Outcome: Also checks your model on new and unseen data.

To better visualize model TensorFlow provides Tensorboard. It helps us to visualize any statistics of the neural network, debug and optimize them. You can check what happens in the code and will give you a detailed understanding of the inner working. You can fix problems very easily with the help of this tool.

3.6 FEASIBILITY STUDY

The feasibility of a music genre classification project depends on several factors, including available resources, data availability, technical challenges, and the project's goals. Here are some considerations regarding feasibility:

1. **Data Availability:** Access to a diverse and sufficiently large dataset is crucial for training a robust classification model. If such a dataset is readily available or can be obtained through partnerships or collaborations, it enhances the feasibility of the project.
2. **Feature Extraction:** Extracting relevant features from audio and, if applicable, visual content requires expertise in signal processing and computer vision. However, with the availability of libraries and frameworks like Librosa for audio processing and OpenCV for visual analysis, this aspect is manageable.
3. **Model Complexity:** Building and training classification models, especially deep learning architectures, can be computationally intensive. The feasibility depends on the computational resources available for training, such as GPUs or cloud computing services.
4. **Labeling Effort:** Manually labeling a large dataset can be time-consuming and resource-intensive. However, if labeled datasets are available or if labeling efforts can be streamlined through semi-supervised or active learning approaches, it enhances feasibility.
5. **Algorithm Selection:** Choosing appropriate algorithms and architectures tailored to the task at hand is crucial. While complex models may offer better performance, simpler models may suffice depending on the project's requirements and constraints.
6. **Evaluation Metrics:** Defining appropriate evaluation metrics and benchmarks for assessing model performance is essential for gauging feasibility and progress throughout the project.
7. **Interdisciplinary Expertise:** Music genre classification projects often require expertise in signal processing, machine learning, and possibly audiovisual analysis. Collaboration between experts in these domains can enhance the feasibility and success of the project.

3.6.1 TECHNICAL FEASIBILITY

Assessing the technical feasibility of a music genre classification project involves evaluating whether the necessary technology, tools, and methods exist to accomplish the task effectively. Here are some aspects to consider:

1. **Data Availability:** The availability of labeled datasets containing audio samples across different genres is essential. Several publicly available datasets, such as GTZAN, Million Song Dataset, and FMA, provide a good starting point.
2. **Feature Extraction:** Techniques for extracting relevant features from audio signals, such as Mel-frequency cepstral coefficients (MFCCs), spectral features, rhythm patterns, and chroma features, are well-established.
3. **Model Selection:** Various machine learning and deep learning models can be applied to music genre classification. Common approaches include Support Vector Machines (SVM), Random Forests, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and hybrid models.
4. **Training Infrastructure:** Training classification models, especially deep learning architectures, may require significant computational resources, such as GPUs or TPUs. Cloud computing platforms like AWS, Google Cloud Platform, or Azure offer scalable infrastructure for model training.
5. **Software Tools:** Libraries and frameworks like TensorFlow, PyTorch, scikit-learn, and Keras provide comprehensive support for building, training, and evaluating machine learning and deep learning models. These tools streamline the development process and facilitate experimentation with different algorithms and architectures.
6. **Evaluation Metrics:** Defining appropriate evaluation metrics for assessing model performance is crucial. Metrics such as accuracy, precision, recall, F1-score, and confusion matrices are commonly used for classification tasks.
7. **Cross-Validation:** Techniques like k-fold cross-validation ensure robust evaluation of model performance and generalization to unseen data. Implementing cross-validation allows for more reliable assessments of model effectiveness.
8. **Deployment:** Once a model is trained and validated, deploying it into production environments requires considerations for scalability, latency, and resource constraints. Frameworks like TensorFlow Serving or containerization tools like Docker facilitate model deployment and serving.

3.6.2 OPERATIONAL FEASIBILITY

Operational feasibility for a music genre classification project involves assessing whether the proposed solution can be effectively integrated into existing operational processes or workflows. Here's how you might evaluate operational feasibility:

1. **User Acceptance:** Determine whether end-users, such as music enthusiasts, streaming platforms, or music recommendation systems, would find value in automated music genre classification. Conduct surveys or user interviews to understand user needs and preferences.
2. **Integration with Existing Systems:** Assess how the classification system would integrate with existing music-related systems or platforms. Compatibility with APIs, data formats, and protocols used by these systems is crucial for seamless integration.
3. **Scalability:** Consider whether the classification system can scale to handle large volumes of music data efficiently. As the volume of music content grows, the system should be able to accommodate increased processing demands without significant performance degradation.
4. **Real-time Processing:** Evaluate whether real-time classification of music genres is necessary for the intended application. For applications like live music streaming or dynamic playlist generation, real-time processing capabilities may be essential.
5. **Training and Maintenance:** Determine the feasibility of continuously updating and retraining the classification model to adapt to evolving music trends and preferences. Establish processes for collecting new data, retraining the model, and deploying updated versions.
6. **User Interface and Experience:** Assess the usability and intuitiveness of the user interface for interacting with the classification system. A user-friendly interface that provides clear feedback and instructions can enhance user adoption and satisfaction.
7. **Training and Support:** Consider the need for training end-users on how to use the classification system effectively. Providing documentation, tutorials, or support channels can help users overcome any challenges they encounter.
8. **Cost Considerations:** Evaluate the costs associated with developing, deploying, and maintaining the classification system. This includes expenses related to data acquisition, infrastructure, software licenses, and ongoing support.

3.6.3 ECONOMIC FEASIBILITY

Assessing the economic feasibility of a music genre classification project involves evaluating whether the benefits of the project outweigh the costs incurred during its development, deployment, and operation. Here are some factors to consider:

1. **Cost of Data Acquisition:** Consider the cost of acquiring labeled music datasets for training the classification model. This may involve purchasing datasets, licensing data from third-party providers, or investing in manual labeling efforts.
2. **Development Costs:** Estimate the expenses associated with developing the classification system, including software development, feature engineering, model training, and testing. This encompasses the cost of hiring developers, data scientists, and domain experts, as well as the cost of computing resources and software tools.
3. **Infrastructure Costs:** Assess the cost of infrastructure required for training and deploying the classification model. This includes the cost of hardware (e.g., servers, GPUs) and cloud computing services (e.g., AWS, Azure) needed to support model training, inference, and scaling.
4. **Maintenance Costs:** Factor in the ongoing costs of maintaining and updating the classification system. This includes expenses related to data updates, model retraining, software updates, and technical support.
5. **Operational Costs:** Consider the costs associated with operating the classification system on a day-to-day basis. This includes expenses such as electricity, network bandwidth, and personnel required for system monitoring, management, and user support.
6. **Revenue Generation:** Explore potential revenue streams associated with the classification system. For example, if the system is integrated into a music streaming platform, it could lead to increased user engagement, retention, and subscription revenues.
7. **Cost Savings or Efficiency Gains:** Evaluate whether the classification system can lead to cost savings or efficiency gains compared to existing manual or semi-automated processes. For example, automating music genre classification could reduce the need for manual tagging of music tracks, saving time and labor costs.

8. **Market Demand and Competitive Landscape:** Assess the market demand for music genre classification solutions and analyze the competitive landscape. Determine whether there is a market need for the proposed solution and whether it offers unique features or advantages compared to existing solutions.
9. **Return on Investment (ROI):** Calculate the expected ROI of the project by comparing the projected benefits (e.g., revenue generation, cost savings) with the total investment (e.g., development costs, operational expenses). Determine whether the expected ROI justifies the investment in the project.

By conducting a thorough economic feasibility analysis, you can determine whether a music genre classification project is financially viable and whether it aligns with your organization's strategic goals and objectives.

User Acceptance:

- Evaluate user acceptance and usability of the classification system. Conduct user surveys or interviews to gather feedback on user needs, preferences, and expectations.
- Assess the feasibility of providing user training and support to facilitate adoption and usage of the system.
- By thoroughly examining these aspects, you can determine the overall feasibility of developing and implementing a music genre classification system and identify any potential challenges or limitations that need to be addressed.

3.7 Data preprocessing:

Data preprocessing was the first step in the project process. For the project, the GTZAN Dataset was used, which consisted of 1000 audio files of 30 seconds each, each file belonging to one of ten different music genres. Preprocessing the dataset included converting the audio signals to spectrograms and dividing them into 128x128 pixel images. The dataset was then divided into training and testing sets, with 80% of the data being used for training and 20% for testing.

MFCC: It stands for Mel-Frequency Cepstral Coefficients. The MFCC algorithm converts the spectral information of an audio signal into a set of coefficients. These coefficients are used to characterize the signal's spectral envelope. Figure 6 shows MFCC coefficients with time on x-axis and MFCC index on y-axis. The color represents the magnitude of the coefficient.

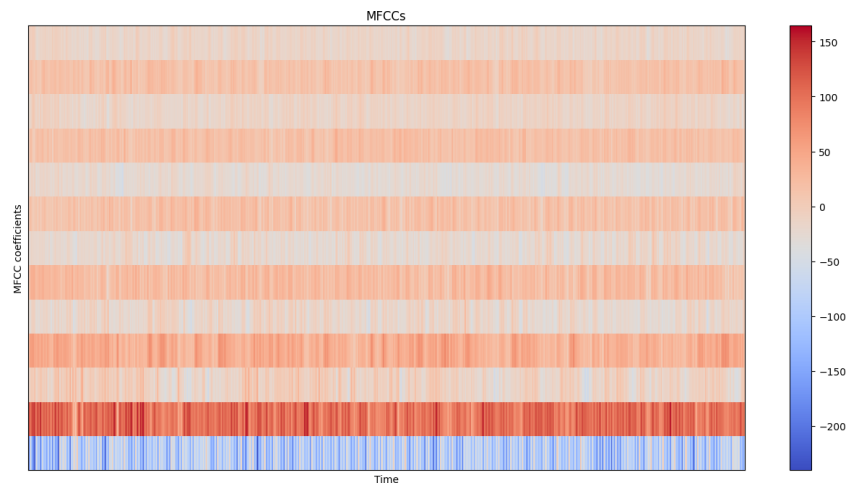


Fig. 4. Mel-spectrogram visualization

DESIGN

4. DESIGN

4.1 Class Diagram

User: Represents the user interacting with the system.

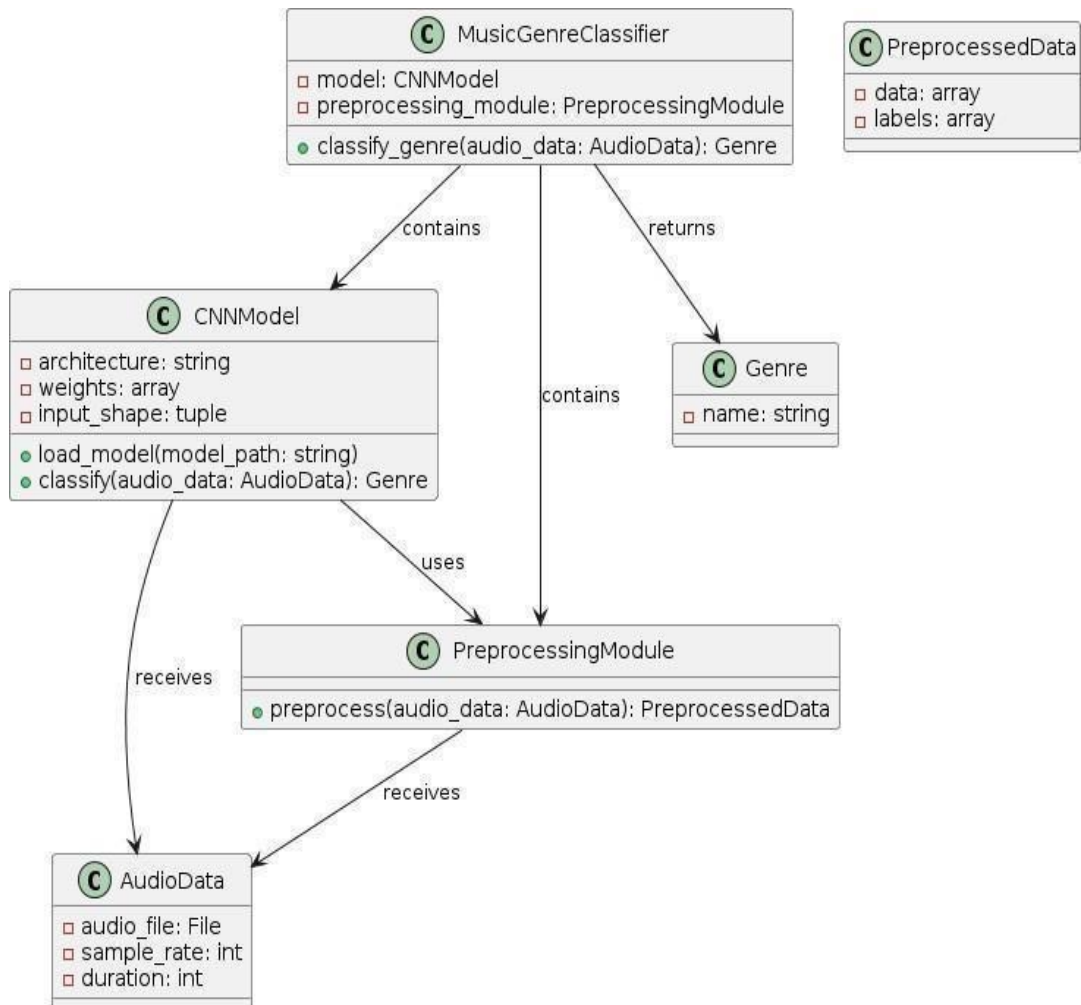
Audio File: Represents an audio file with its path and data, responsible for loading audio data from a file.

Preprocessor: Handles preprocessing of audio data.

Feature Extractor: Extracts features from preprocessed audio data.

Genre: Represents a predicted genre with its name and confidence.

This diagram illustrates the classes involved in the music genre classification system and their relationships.

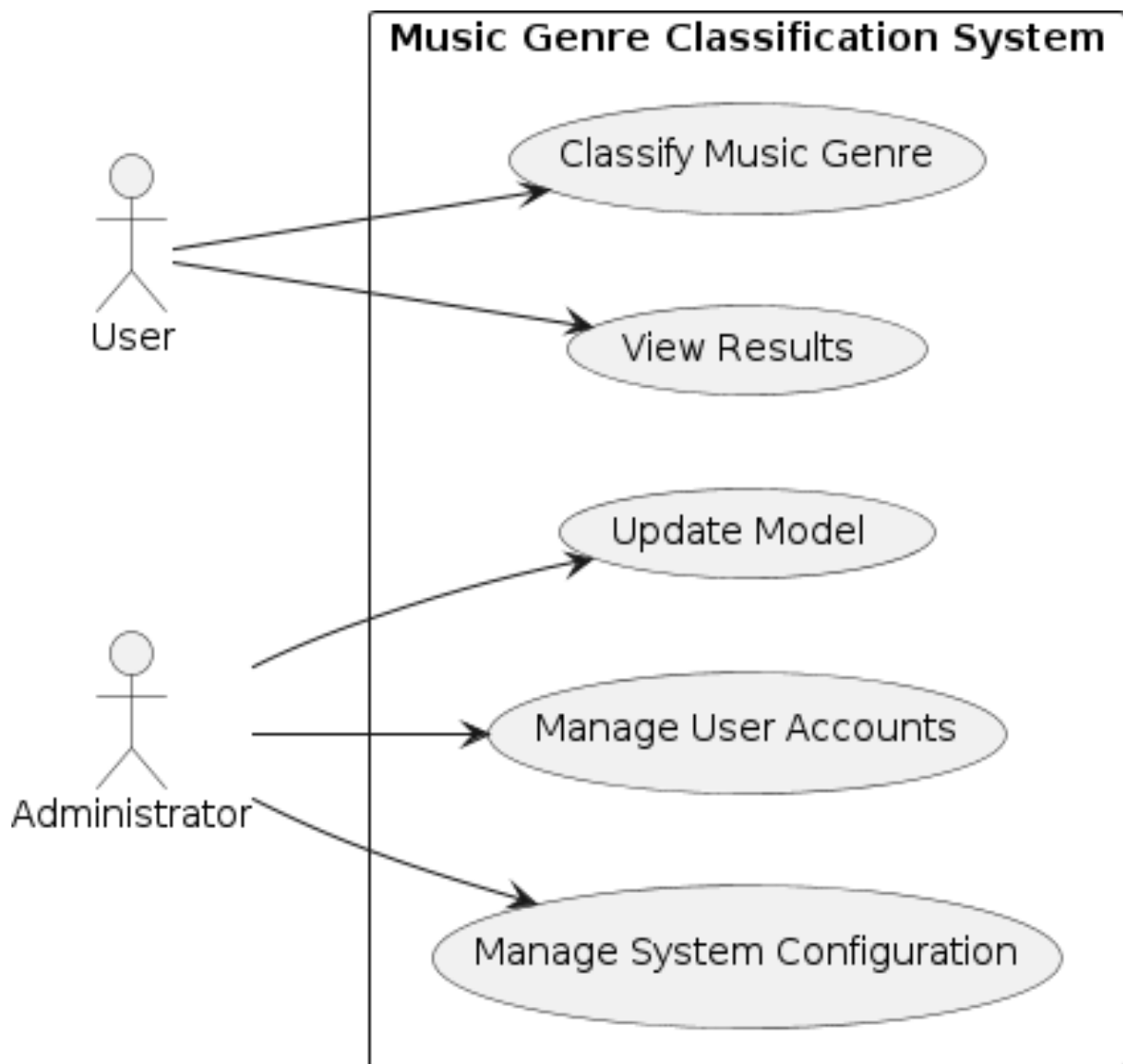


4.2 Use Case Diagram

In this use case diagram:

- **User:** Represents the user interacting with the system.
- **Provide Audio File:** Use case where the user provides an audio file for genre classification.
- **Receive Genre Prediction:** Use case where the user receives the predicted genre after classification.

This diagram illustrates the interactions between the user and the system in the context of music genre classification using deep learning with CNNs.

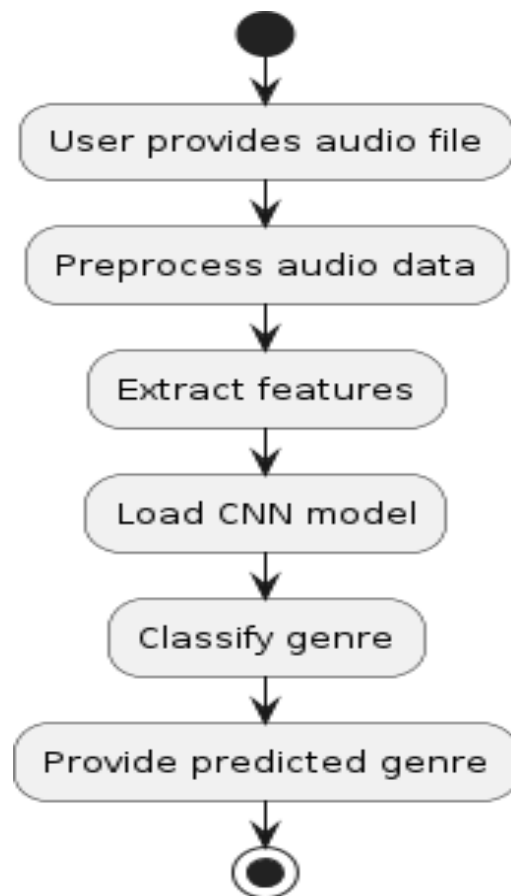


4.3 Activity Diagram

In this activity diagram:

- **Start:** Indicates the start of the process.
- **User provides audio file:** Represents the user providing an audio file for genre classification.
- **Preprocess audio data:** Preprocesses the audio data, which may involve tasks like noise reduction and normalization.
- **Extract features:** Extracts features from the preprocessed audio data.
- **Load CNN model:** Loads the CNN model used for genre classification.
- **Classify genre:** Uses the CNN model to classify the genre based on the extracted features.
- **Provide predicted genre:** Provides the predicted genre to the user.
- **Stop:** Indicates the end of the process.

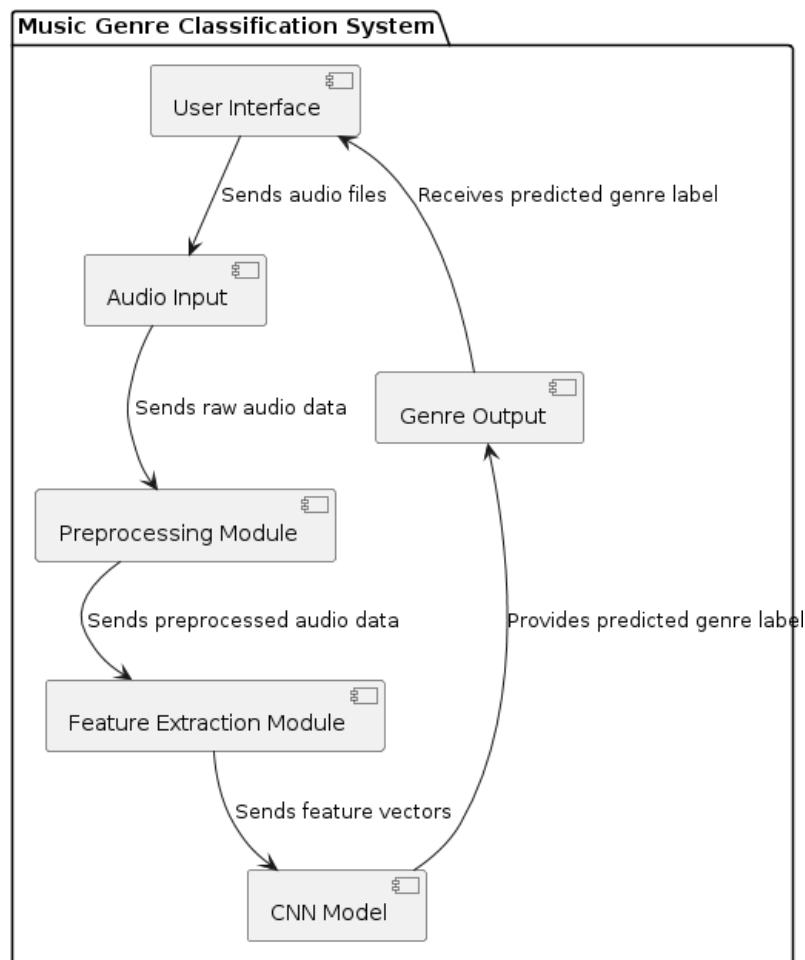
This diagram visualizes the sequence of steps involved in the music genre classification process using deep learning with CNNs.



4.4 Component Diagram

In this diagram:

- **User Interface:** Represents the interface through which users interact with the system, such as providing audio files and receiving predicted genre labels.
- **Audio Input:** Receives the audio files from the user and forwards them to the preprocessing module.
- **Preprocessing Module:** Handles the preprocessing of audio data, such as removing noise or normalizing the audio.
- **Feature Extraction Module:** Extracts relevant features from the preprocessed audio data, such as spectrograms or MFCCs.
- **CNN Model:** Performs the classification of music genre based on the extracted features using Convolutional Neural Networks.
- **Genre Output:** Provides the predicted genre label back to the user interface.



IMPLEMENTATION

5. Implementation

```
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/drive')
example_file="/content/drive/MyDrive/genres-20231105T184718Z
001/genres/disco/disco.00099.wav"
#read

import soundfile as sf
signal, sample_rate = sf.read(example_file)

#load audio file with Librosa
signal,sample_rate = librosa.load(example_file)

print(signal)

print(sample_rate)
FIG_SIZE =(16,8)

# Create a time arrayfor the waveform chatgpt
time = np.arange(0, len(signal)) / sample_rate

# Create the waveform plot
plt.figure(figsize=(10, 4))
plt.plot(time, signal, lw=1)
plt.title('Waveform Plot')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show()

#WAVEFORM
plt.figure(figsize=FIG_SIZE)
librosa.display.waveshow(signal,sr=sample_rate,alpha=0.5)
plt.xlabel("Time (S)")
plt.ylabel("Amplitude")
plt.title("WaveForm")

#Fourier Transform (FFT) -Frequency Domain
fft = np.fft.fft(signal)
#Calculate the Magnitude (abs values on complex numbers)
spectrum =np.abs(fft)
#Create the frequency variable
f=np.linspace(0, sample_rate , len(spectrum))
```

```

#Plot Spectrum
plt.figure(figsize=FIG_SIZE)
plt.plot(f,spectrum,alpha=0.5)
plt.xlabel("Frequency")
plt.ylabel("Magnitude")
plt.title("Power Spectrum")

#Take half of the spectrum and frequency
left_spectrum=spectrum[:int(len(spectrum)/2)]
left_f = f[:int(len(spectrum)/2)]

#Plot Spectrum
plt.figure(figsize=FIG_SIZE)
plt.plot(left_f,left_spectrum,alpha=0.5)
plt.xlabel("Frequency")
plt.ylabel("Magnitude")
plt.title("Power Spectrum")

#Spectrogram (STFT)
hop_length = 512 #no.of samples
n_fft = 2048 #no.of samples for window

#Perform STFT
stft = librosa.stft(signal,n_fft=n_fft,hop_length=hop_length)

#Calculate the Magnitude (abs values on complex numbers)
spectrogram= np.abs(stft)

#Plot the spectrogram
plt.figure(figsize=FIG_SIZE)
librosa.display.specshow(spectrogram,sr=sample_rate,hop_length=hop_length)
plt.xlabel("Time")
plt.ylabel("Frequency")
plt.colorbar()
plt.title("Spectrogram")

#Apply Logarithm to get values in Decimals
log_spectrogram = librosa.amplitude_to_db(spectrogram)

#Plot the Spectrogram in Decibels
plt.figure(figsize=FIG_SIZE)
librosa.display.specshow(log_spectrogram, sr=sample_rate, hop_length=hop_length)
plt.xlabel("Time")
plt.ylabel("Frequency")
plt.colorbar(format = "%+2.0f dB")

```

```

plt.title("Spectrogram (dB)")

#MFCCs (we use 13 MFCCs) chatgpt
MFCCs = librosa.feature.mfcc(y=signal, sr=sample_rate, n_fft=n_fft,
hop_length=hop_length, n_mfcc=13)

#Plot MFCCs
plt.figure(figsize=FIG_SIZE)
librosa.display.specshow(MFCCs,sr=sample_rate,hop_length=hop_length)
plt.xlabel("Time")
plt.ylabel("MFCC coefficients")
plt.colorbar()
plt.title("MFCCs")

import json
import os
import math

DATASET_PATH="/content/drive/MyDrive"
JSON_PATH = "metal.00099.wav"
SAMPLE_RATE = 22050
TRACK_DURATION = 30 # measured in seconds
SAMPLES_PER_TRACK = SAMPLE_RATE * TRACK_DURATION

def
save_mfcc(dataset_path,json_path,num_mfcc=13,n_fft=2048,hop_length=512,num_seg
ments=5):
    #dictionary to store mapping,labels and MFCCs
    data={
        "mapping":[],
        "labels":[],
        "mfcc":[]
    }

    samples_per_segment = int(SAMPLES_PER_TRACK/num_segments)
    num_mfcc_vectors_per_segment = math.ceil(samples_per_segment /hop_length)

    #loop through all genre sub-folder
    for i, (dirpath,dirnames,filenames) in enumerate(os.walk(dataset_path)):

        #ensure we're processing a genre sub-folder level
        if dirpath is not dataset_path:

            #save genre label in the mapping
            semantic_label = dirpath.split("/")[-1]

```

```

data["mapping"].append(semantic_label)
print("\nProcessing: {}".format(semantic_label))

#process all audio files in genre sub-dir
for f in filenames:
    #load audio file
    file_path=os.path.join(dirpath,f)
    signal,sample_rate = librosa.load(file_path,sr=SAMPLE_RATE)

    #process all segments of audio file
    for d in range(num_segments):
        #calculate start and finish sample for current segment
        start = samples_per_segment *d
        finish = start + samples_per_segment

        #extract mfcc
        mfcc =
librosa.feature.mfcc(signal[start:finish],sample_rate,n_mfcc=num_mfcc,n_fft=n_fft,
hop_length=hop_length)
        mfcc = mfcc.T

        #store onlymfcc feature with expected number of vectors
        if len(mfcc) == num_mfcc_vectors_per_segment:
            data["mfcc"].append(mfcc.tolist())
            data["labels"].append(i-1)
            print("{} , segments: {}".format(file_path, d+1))
    with open(json_path, "w") as fp:
        json.dump(data,fp,indent=4)

```

Part-2:

```

import random, os, glob # default python modules that let me randomise and manipulate
files
import numpy as np # for data manipulation through arrays
import tensorflow as tf
from keras.models import Sequential # the model I will use
from keras.layers import Dropout, Dense, Conv2D, MaxPool2D, Flatten, Reshape,
BatchNormalization, GlobalAveragePooling2D # layers I will incorporate
from keras.callbacks import EarlyStopping # for better training
from tensorflow.keras.applications import VGG19 # transfer learning model
from keras import backend
from livelossplot import PlotLossesKeras # to visually displayhow mymodel improves
as training progresses
import librosa # to demonstrate the creation of a mel spectrogram
from librosa.display import specshow

```

```

import matplotlib.pyplot as plt
import IPython.display as ipd
from sklearn.metrics import confusion_matrix
import seaborn as sns

def setRandom():
    seed = 0 # random seed value
    os.environ["PYTHONHASHSEED"] = str(seed) # if this is not set, a random value is
    used to seed the hashes of some objects
    random.seed(seed) # sets the base python and numpy random seeds
    np.random.seed(seed)
    tf.random.set_seed(seed) # sets the tensorflow random seed
    tf.compat.v1.set_random_seed(seed)

from google.colab import drive
drive.mount('/content/drive')

# an example file
filePath="/content/drive/MyDrive/genres_original/disco/disco.00003.wav"

file, samplingRate = librosa.load(filePath)
example, _ = librosa.effects.trim(file)

hopLength = 512 # the number of samples between successive columns of the
spectrogram

spectrogram = librosa.power_to_db(librosa.feature.melspectrogram(y = example, sr =
samplingRate, n_fft = 2048, hop_length = hopLength, n_mels = 128, power = 4.0), ref =
np.max)

plt.figure(figsize = (3, 2))
librosa.display.specshow(spectrogram, sr = samplingRate, hop_length = hopLength,
x_axis = "off", y_axis = "off")
ipd.Audio(example, rate = samplingRate)

source = "/content/drive/MyDrive/images_original" # source folder path
genres = ["blues", "classical", "country", "disco", "hiphop", "jazz", "metal", "pop",
"reggae", "rock"] # list with the genre folder names

for genre in genres: # iterate through each genre folder
    path = os.path.join(source, genre)
    if not os.path.isdir(path):
        print(f'Skipping {genre} folder as it does not exist.')
        continue
    pngs = [i for i in os.listdir(path) if i[-4:] == ".png"] # get a list of .png files in the genre

```

```

folder
    print(f"Size of {genre} dataset: {len(pngs)} files.")

setRandom()
split = [80, 9, 10]
train, val, test = {}, {}, {} # empty dictionaries to store the filepaths
trainLen, valLen, testLen = {}, {}, {} # empty dictionaries to store the number of files
under each genre for each dataset
dictionaries = [train, val, test]

for d in dictionaries:
    if d == train: num = slice(0, split[0])
    elif d == val: num = slice(split[0], split[0] + split[1])
    else: num = slice(split[0] + split[1], split[0] + split[1] + split[2])
    for genre in genres: # iterate through each genre folder
        path = os.path.join(source, genre)
        pngs = glob.glob(os.path.join(path, "*.png")) # get a list of .png filepaths in the
genre folder
        selected = pngs[num] # take the first 80 files
        d[genre] = selected # store the selected files in the dictionary

lenDictionaries = [{genre: len(d[genre]) for genre in genres} for d in dictionaries]
print(f"\033[1mTraining:\033[0m {lenDictionaries[0]}")
print(f"\033[1mValidation:\033[0m {lenDictionaries[1]}")
print(f"\033[1mTest:\033[0m {lenDictionaries[2]}")

batchSize = 32 # typical batch size for a neural network
genreMap = {
    "blues": 0,
    "classical": 1,
    "country": 2,
    "disco": 3,
    "hiphop": 4,
    "jazz": 5,
    "metal": 6,
    "pop": 7,
    "reggae": 8,
    "rock": 9
}
inverseGenreMap = {value: key for key, value in genreMap.items()}

def createDataset(d):
    imgSize = (288, 432) # define image and batch parameters
    imageList, labelList = [], [] # create lists to store images and labels
    for genre, paths in d.items():

```

```

    for path in paths:
        image = tf.cast(tf.image.resize(tf.image.decode_png(tf.io.read_file(path),
channels = 3), imgSize), tf.float32) / 255.0 # normalise pixel values between 0 and 1
(preprocessing!)
        imageList.append(image)
        labelList.append(genreMap[genre]) # convert genre to its integer label

    dataset = tf.data.Dataset.from_tensor_slices((imageList,
labelList)).shuffle(buffer_size=len(imageList)).batch(batchSize) # create and return
tensorflow dataset
    return(dataset)

def prep(ds):
    out = (
        ds.map(lambda image, label: (tf.image.convert_image_dtype(image, tf.float32),
label)) # modifies the image tensor's data type to floats
        .cache() # cache dataset elements in memory or on disk to speed up data loading
        .prefetch(buffer_size = tf.data.experimental.AUTOTUNE) # prefetch dataset
elements in the background and automatically optimise data loading
        )
    return out # return the prepared and optimised dataset

training, validation, testing = prep(createDataset(train)), prep(createDataset(val)),
prep(createDataset(test))
print("Datasets created.") # to let us know when it's finished running (:

def view_dataset(dataset):
    genreExamples = { } # dictionary to store examples for each label

    for images, labels in dataset:
        for image, label in zip(images, labels):
            label = int(label.numpy()) # convert label tensor to integer
            if label not in genreExamples:
                genreExamples[label] = image

    if len(genreExamples) == len(genres):
        break

    # display the randomly chosen examples
    plt.figure(figsize = (30, 20))
    for label, image in genreExamples.items():
        ax = plt.subplot(1, len(genres), label + 1)
        plt.imshow(image)
        plt.title(inverseGenreMap[label])
        plt.axis("off")

```

```

plt.show()

print("\033[1mTraining Examples:\033[0m"); view_dataset(training) # shows a labelled
example of a mel spectrogram from each genre from each dataset
print("\033[1mValidation Examples:\033[0m"); view_dataset(validation)
print("\033[1mTesting Examples:\033[0m"); view_dataset(testing)

inputShape = [288, 432, 3] # the shape of the images (288px tall, 432px wide, and 3
colour channels/RGB)
earlyStopping = EarlyStopping( # a custom earlystopping setup to automatically stop
training when loss doesn't increase enough after a number of epochs
    min_delta = 0.001, # minimum amount of change to count as an improvement
    patience = 20, # how many epochs to wait before stopping
    restore_best_weights = True # tells it to restore back to when loss was at its lowest
value
)
from tensorflow.keras.applications import InceptionV3 # transfer learning model
baseModel = InceptionV3(input_shape = inputShape, weights = "imagenet", include_top
= False, pooling = "avg")

for layer in baseModel.layers:
    layer.trainable = False # freeze the pre-trained layers

transfer = Sequential([
    baseModel,

    Flatten(),
    BatchNormalization(),
    Dense(512, activation = "relu"),
    Dropout(0.3),
    Dense(256, activation = "relu"),
    Dropout(0.3), # dropout layer to prevent overfitting
    Dense(128, activation = "relu"),
    Dropout(0.3),
    Dense(len(genres), activation = "softmax")
])

transfer.compile(optimizer = tf.keras.optimizers.SGD(lr = 0.0001), loss =
"sparse_categorical_crossentropy", metrics = ["accuracy"])
transfer.summary()

cnn = Sequential([
    BatchNormalization(input_shape = inputShape),

    Conv2D(32, (3, 3), activation = "relu"),

```

```

MaxPool2D((2, 2)),

Conv2D(64, (3, 3), activation = "relu"),
MaxPool2D((2, 2)),

Conv2D(128, (3, 3), activation = "relu"),
MaxPool2D((2, 2)),

Conv2D(256, (3, 3), activation = "relu"),
MaxPool2D((2, 2)),

Conv2D(512, (3, 3), activation = "relu"),
MaxPool2D((2, 2)),

Flatten(),
Dense(1024, activation = "relu"),
Dropout(0.5),
Dense(512, activation = "relu"),
Dropout(0.5),
BatchNormalization(),
Dense(len(genres), activation = "softmax")
])

cnn.compile(optimizer = tf.keras.optimizers.SGD(lr = 0.001), loss =
"sparse_categorical_crossentropy", metrics = ["accuracy"]) #
"sparse_categorical_crossentropy" because labels are integers
cnn.summary()

def confusionMatrix(model, name):
    trueLabels = np.concatenate([y for x, y in testing], axis = 0) # get the true labels from
the testing dataset

    predictedLabels = np.argmax(model.predict(testing, verbose = 0), axis = 1) # get the
predicted labels from the model

    matrix = confusion_matrix(trueLabels, predictedLabels) # create the confusion matrix

    plt.figure() # plot the confusion matrix using seaborn for the heatmap
    sns.heatmap(matrix, annot = True, cmap = "Greens", xticklabels = genres, yticklabels
= genres)
    plt.xlabel("Predicted Genre")
    plt.ylabel("True Genre")
    plt.title(f'{name} Model: Confusion Matrix')
    plt.show()

```

```

trainStats, valStats, testStats = model.evaluate(training, verbose = 0),
model.evaluate(validation, verbose = 0), model.evaluate(testing, verbose = 0)
print(f"\033[1m{name} Model\033[0m")
print(f"Training Accuracy: {100-(round(trainStats[1] * 100, 4))}% \nTrain Loss:
{round(trainStats[0], 4)}\n")
print(f"Validation Accuracy: {100-(round(valStats[1] * 100, 4))}% \nTest Loss:
{round(valStats[0], 4)}\n")
print(f"Testing Accuracy: {100-(round(testStats[1] * 100, 4))}% \nTest Loss:
{round(testStats[0], 4)}")

confusionMatrix(transfer, "Transfer")
confusionMatrix(cnn, "Custom CNN")

setRandom()
genre, paths = random.choice(list(test.items())) # chooses a random genre and
corresponding list of paths
path = paths[random.randint(0, len(paths) - 1)] # chooses a random path within that list
soundPath = f"{path[:-4]}.wav".replace("images_original", "genres_original") # creates a
new variable for the image path's respective audio file path
soundPath = soundPath[:-9] + "." + soundPath[-9:]

image = tf.image.decode_png(tf.io.read_file(path), channels = 3)
image = tf.image.resize(image, inputShape[:2]) # resize the image to the input shape of
the model
image = tf.cast(image, tf.float32) / 255.0 # normalise pixel values
image = np.expand_dims(image, axis = 0) # expand dimensions to match the batch size
(even though it's just one image)

predictions = cnn.predict(image) # make a prediction using the custom cnn model
predictedGenre = inverseGenreMap[np.argmax(predictions)] # get the predicted class
label
print(f"\033[1mPredicted genre:\033[0m {genre}")
print(f"\033[1mActual genre:\033[0m {genre}")
print(f"\033[1mFilepath:\033[0m {soundPath[71:]}")
ipd.Audio(soundPath)

```

RESULT ANALYSIS

6. RESULT ANALYSIS

Convolutional Neural Network (CNN):

To build a Convolutional Neural Network (CNN) model for music genre classification, train the model on a training set, evaluate its performance on a validation set, and then report the model's accuracy.

```
history = model_cnn.fit(x_train,y_train,validation_data=(x_validation,y_validation),batch_size=32,epochs=50)
```

Epoch 48/50

188/188 [=====] - 13s 68ms/step - loss: 0.4931 - accuracy: 0.8301 - val_loss: 0.7348 - val_accuracy: 0.7483

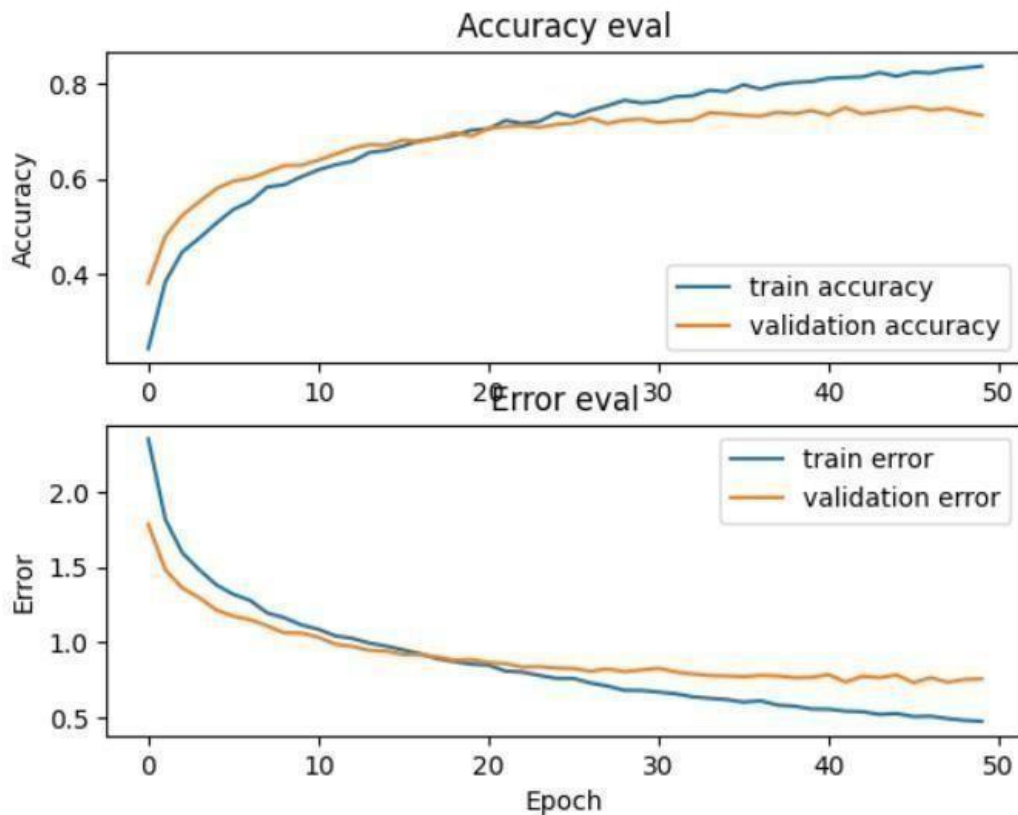
Epoch 49/50

188/188 [=====] - 12s 62ms/step - loss: 0.4812 - accuracy: 0.8332 - val_loss: 0.7534 - val_accuracy: 0.7403

Epoch 50/50

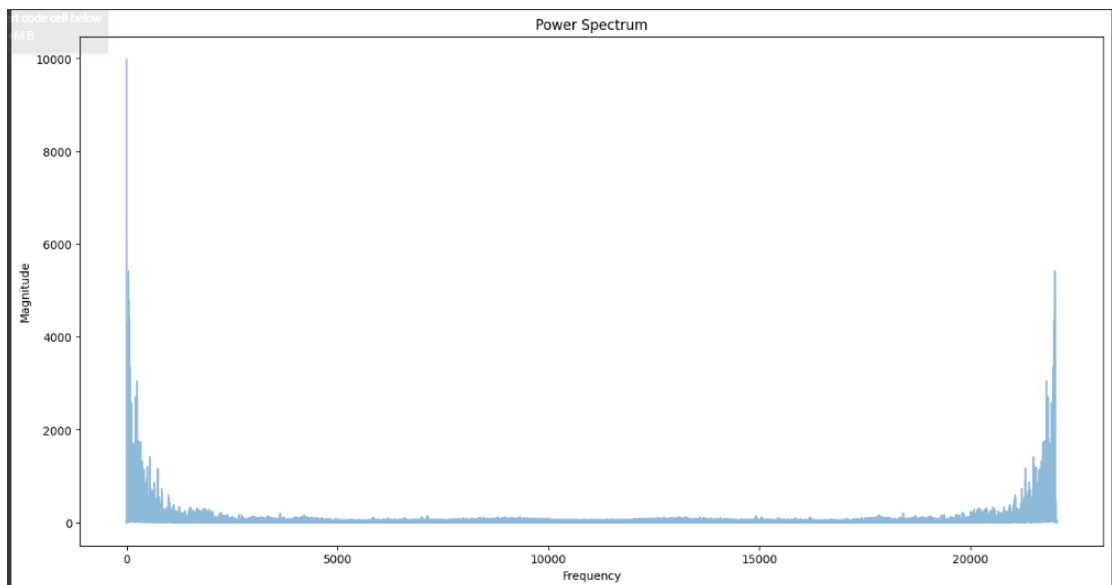
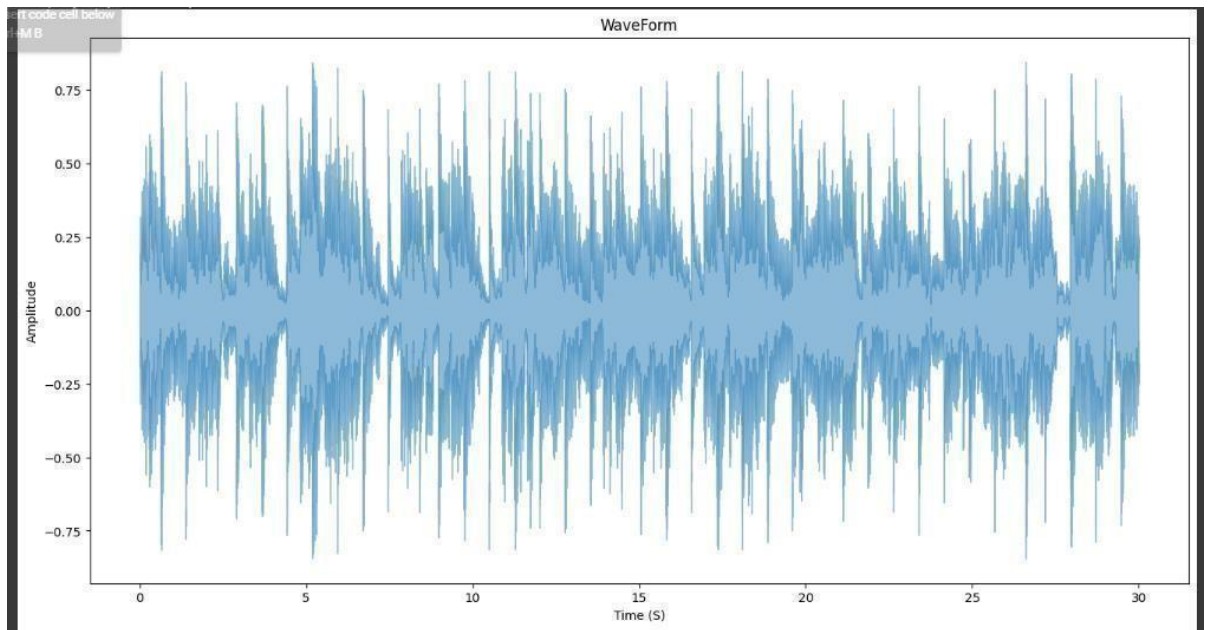
188/188 [=====] - 11s 59ms/step - loss: 0.4751 - accuracy: 0.8371 - val_loss: 0.7569 - val_accuracy: 0.7336

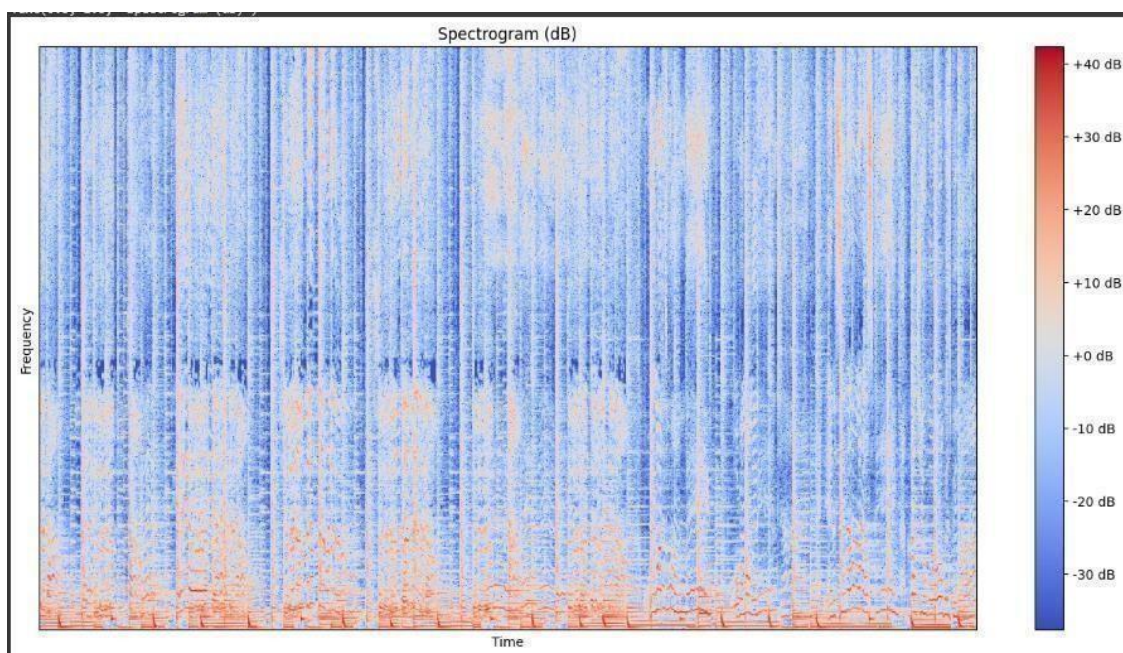
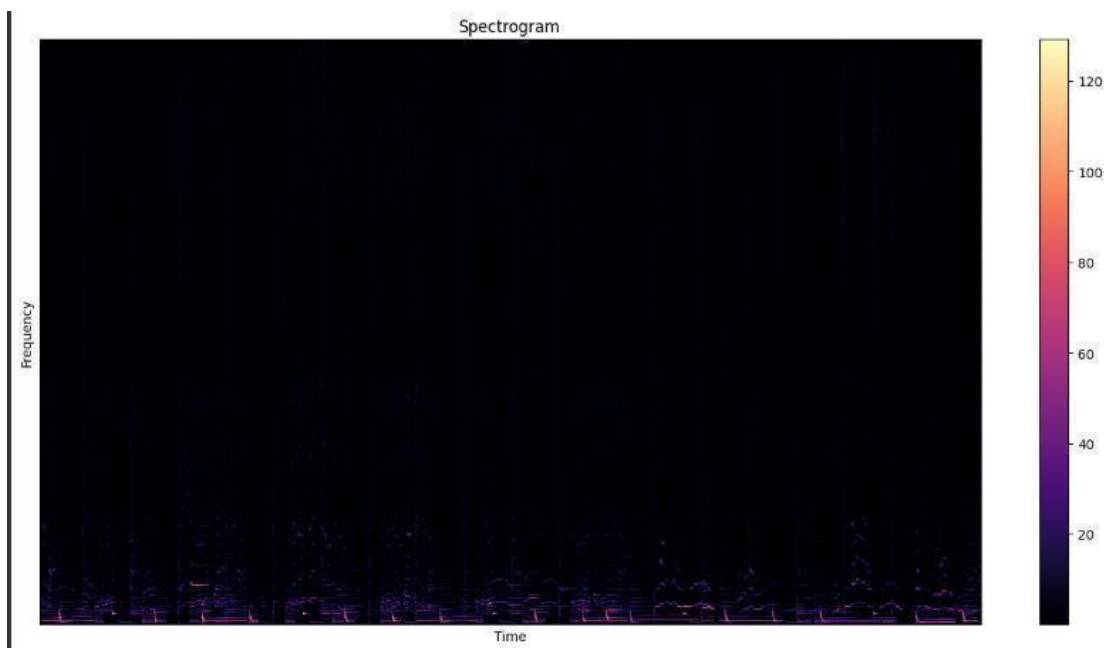
```
#plot accuracy and error as a function of the epochs  
plot_history(history)
```

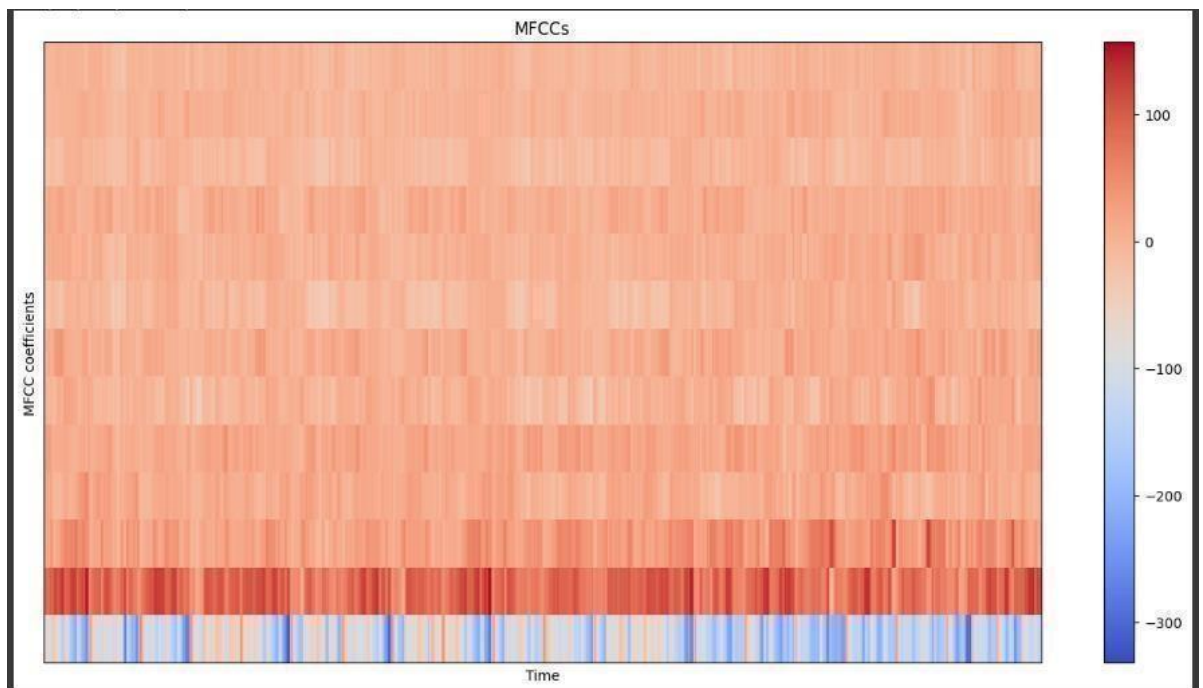


OUTPUT SCREENSHOTS

7. OUTPUT SCREENSHOTS







```
# an example file
filePath = "/content/drive/MyDrive/genres_original/disco/disco.00003.wav"

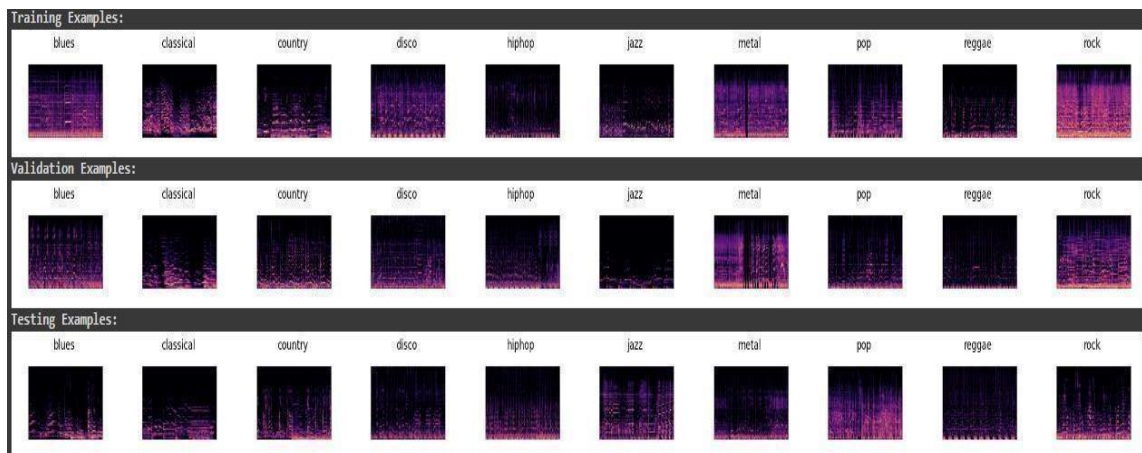
file, samplingRate = librosa.load(filePath)
example, _ = librosa.effects.trim(file)

hopLength = 512 # the number of samples between successive columns of the spectrogram

spectrogram = librosa.power_to_db(librosa.feature.melspectrogram(y = example, sr = samplingRate, n_fft = 2048, hop_length = hopLength, n_mels = 128, power = 4.0), ref = np.max)

plt.figure(figsize = (3, 2))
librosa.display.specshow(spectrogram, sr = samplingRate, hop_length = hopLength, x_axis = "off", y_axis = "off")
ipd.Audio(example, rate = samplingRate)
```

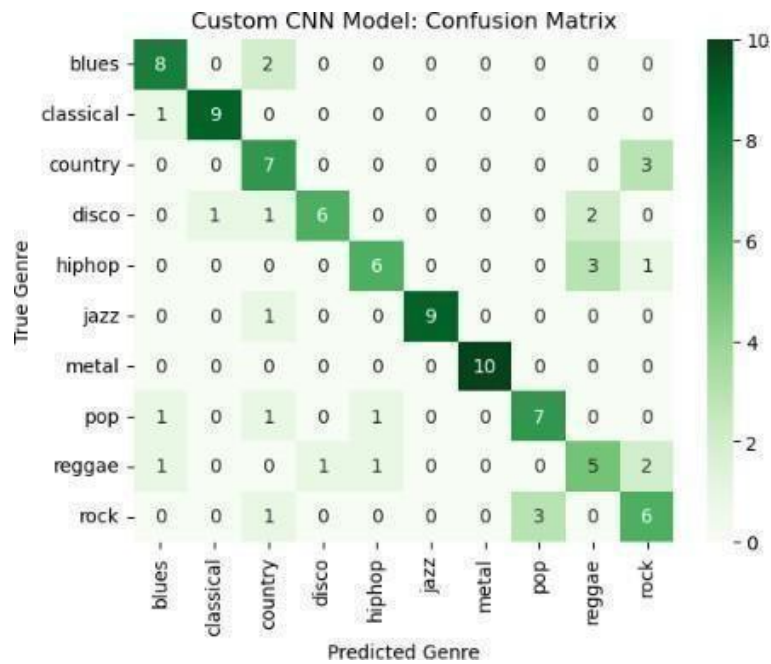
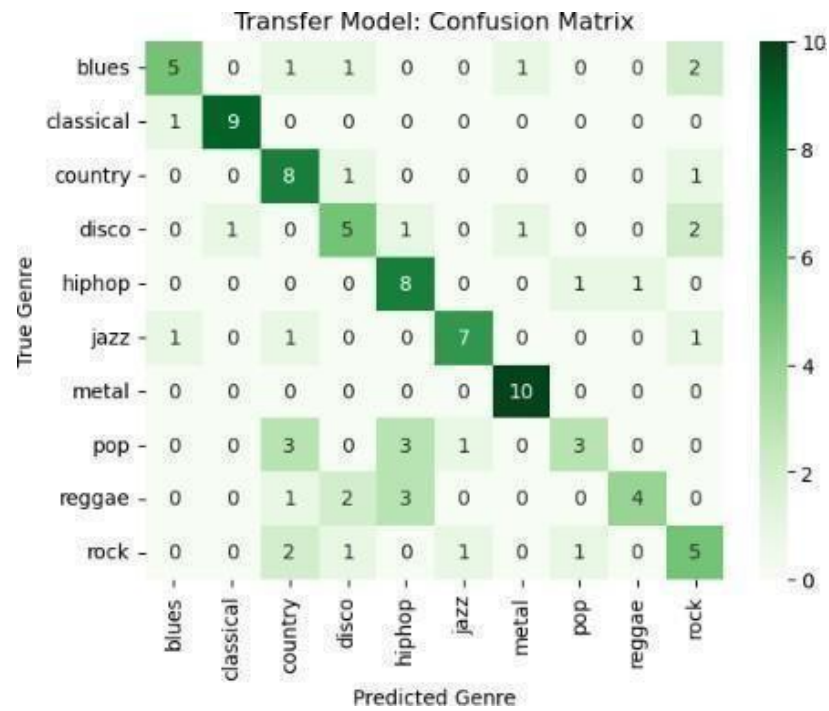
0:00 / 0:30



```
Size of blues dataset: 100 files.  
Size of classical dataset: 100 files.  
Size of country dataset: 100 files.  
Size of disco dataset: 100 files.  
Size of hiphop dataset: 100 files.  
Size of jazz dataset: 99 files.  
Size of metal dataset: 100 files.  
Size of pop dataset: 100 files.  
Size of reggae dataset: 100 files.  
Size of rock dataset: 100 files.
```

conv2d_98 (Conv2D)	(None, 14, 23, 512)	1180160
max_pooling2d_8 (MaxPooling2D)	(None, 7, 11, 512)	0
flatten_1 (Flatten)	(None, 39424)	0
dense_4 (Dense)	(None, 1024)	40371200
dropout_3 (Dropout)	(None, 1024)	0
dense_5 (Dense)	(None, 512)	524800
dropout_4 (Dropout)	(None, 512)	0
batch_normalization_96 (Batch Normalization)	(None, 512)	2048
dense_6 (Dense)	(None, 10)	5130
=====		
Total params: 42471766 (162.02 MB)		
Trainable params: 42470736 (162.01 MB)		
Non-trainable params: 1030 (4.02 KB)		

batch_normalization_94 (Batch Normalization)	(None, 2048)	8192
dense (Dense)	(None, 512)	1049088
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1290
=====		
Total params: 23025578 (87.84 MB)		
Trainable params: 1218698 (4.65 MB)		
Non-trainable params: 21806880 (83.19 MB)		



Transfer Model

Training Accuracy: 87.625%

Train Loss: 2.3346

Validation Accuracy: 90.0%

Test Loss: 2.3372

Testing Accuracy: 85.0%

Test Loss: 2.3225

Custom CNN Model

Training Accuracy: 89.375%

Train Loss: 2.3057

Validation Accuracy: 88.8889%

Test Loss: 2.3053

Testing Accuracy: 95.0%

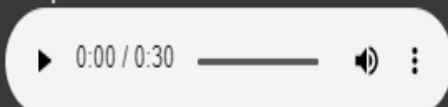
Test Loss: 2.3058

1/1 [=====] - 1s 826ms/step

Predicted genre: metal

Actual genre: metal

Filepath:



CONCLUSION

8. CONCLUSION

The CNN technique is used to meet the automated MGC objectives. The audio files are divided into shorter segments. MFCCs are retrieved from audio segments and saved as JSON files. Each segment contains 13 MFCCs that represent each audio file uniquely. By encoding audio signals in a small feature space, MFCC decreases the dimensionality of the signals. This improves computing efficiency and reduces the complexity of machine learning methods used for audio processing tasks like automated music genre classification. As a result, it is well suited to capture genre-specific elements that are consistent across different pitches and timbres of music. We attained an accuracy of 83%, demonstrating that the model can classify the genre more accurately. The F1 rating is 83%. Plots for accuracy versus epochs and loss versus epochs are shown to provide a more visual understanding of the model's performance. Our model achieved an 83% precision and recall rate. The confusion matrix is presented to summarize the forecasts for each genre. Future enhancements: Current work processes the audio if it is of 30- second duration. Further experiments should be done to handle audio of any length. Implementing music genre classification for various audio formats can be explored. The implemented model works well for the WAV format. There are many other formats such as MP3, FLAC, and others. Data Augmentation can be done to further improve the performance of the model. It helps to increase the diversity and quantity of the training data. Hyperparameter tuning assists in determining the best combination of Hyperparameter values to achieve the best performance.

FUTURE ENHANCEMENT

9. FUTURE ENHANCEMENT

The future scope of music genre classification using deep learning, particularly convolutional neural networks (CNNs), is highly promising. Continued research and development in this area could lead to significant advancements, including improved accuracy in genre classification, making these models more reliable for various applications. Further exploration of CNN architectures could enable the automatic learning of more abstract and meaningful features from audio signals, potentially enhancing classification performance. Optimizing models for real-time classification could pave the way for applications like automatic music tagging and recommendation systems that dynamically respond to user preferences. Additionally, developing models that can accurately classify songs that blend multiple genres or do not fit traditional genre definitions could lead to more comprehensive music understanding. Enhancing the interpretability of deep learning models is also crucial for understanding why a model predicts a certain genre for a particular audio input, which is essential for applications requiring human validation. Integrating other modalities, such as lyrics, album art, or user listening patterns, with audio signals could further enhance genre classification systems. Overall, the future of music genre classification using deep learning CNNs is likely to involve advancements in model architectures, feature learning, interpretability, and integration with other modalities, leading to more accurate, efficient, and versatile systems for music analysis and understanding.

BIBLIOGRAPHY

10. BIBLIOGRAPHY

1. Music Genre Classification with Python <https://towardsdatascience.com/music-genre-classification-withpython- c714d032f0d8>
2. T. Shaikh and A. Jadhav, *Music Genre Classification Using Neural Network*, in Proceedings of the International Conference on Automation, Computing and Communication, ICACC, 05 May 2022, Mumbai, India (2022)
3. A. A. Khamees, H. D. Hejazi, M. Alshurideh, S. A. Salloum, *Classifying Audio Music Genres Using CNN and RNN*, in Proceedings of the Advances in Intelligent Systems and Computing, 5 May 2021, Cairo, Egypt (2021)
4. B. Dave, V. Chavan, M. Khan, Dr. V. Shah, J. Emerg. Tech. Innov. Res **8**, 4 (2021).
5. N. Ndou, R. Ajoodha and A. Jadhav, *Music Genre Classification: A Review of Deep-Learning and Traditional Machine-Learning Approaches*, in Proceedings of the IEEE Aimtronics, 5 May 202, Toronto, Canada (2021)
6. M. Vaibhavi, R. K. Pisipati, J. Inno. Sci. Sust. Tech **1**, 1 (2021)
7. Seethal V, Dr. A. Vijayakumar, Intl. J. Tren. Scien. Res. Devp **5**, 4 (2021)
8. N. Parab, Shikta Das, G. Goda, A. Naik, Intl. Res. J. Engg. Tech **8**, 10 (2021)
9. S. Garg, A. Varshney, Intl. J. Adv. Res. Comp. Com. Engg **11**, 5 (2022).
10. Dias, Jessica, Pillai, Vaishak, Deshmukh, Hrutvik, Shah, Ashok, *Music Genre Classification & Recommendation System using CNN*, SSRN, 4111849 (2022)
11. J. Mehta, D. Gandhi, G. Thakur and P. Kanani, *Music Genre Classification using Transfer Learning on log-based MEL Spectrogram*, in Proceedings of the International Conference on Computing Methodologies and Communication, ICCMC, 6 May 2021, Erode, India (2021)
12. A. Kamala, H. Hassani, Kurdish, *Music Genre Recognition Using a CNN and DNN*, in Proceedings of the International Electronic Conference on Applied Sciences, IECAS, 21–25 November 2022, Universidad San Ignacio de Loyola, Peru (2022).
13. D. S. Lau, R. Ajoodha, *Music Genre Classification: A Comparative Study Between Deep Learning and Traditional Machine Learning Approaches*, in Proceedings of the International Congress on Information and Communication Technology, ICICT, 21-24 February.