MUSIC EMOTION RECOGNITION USING DEEP NEURAL NETWORKS

HAKAN PÜRE

MAY 2024

ÇANKAYA UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

M.Sc. Thesis in

COMPUTER ENGINEERING

MUSIC EMOTION RECOGNITION USING DEEP NEURAL NETWORKS

HAKAN PÜRE

MAY 2024

ABSTRACT

MUSIC EMOTION RECOGNITION USING DEEP NEURAL NETWORKS

PÜRE, HAKAN
M.Sc. in Computer Engineering

Supervisor: Assist. Prof. Dr. Abdül Kadir GÖRÜR
May 2024, 55 pages

Music has an economic value of billions of dollars today. In order to correctly detect the emotion felt in music, correct classification is required. In this study, we try to present a comprehensive research on music emotion recognition (MER) using deep neural networks in order to increase the accuracy of emotion detection and classification from music. Although many studies have been conducted on the music of different countries, there are very few studies on Turkish music. Therefore, we developed our study using a dataset consisting of Turkish songs. In our research, we used deep learning architectures (CNN, LSTM) and machine learning algorithm (RFC) to discover the relationships between various sound features, melody, harmony, rhythm, complex patterns and the emotions triggered by these features. Our first goal in our study was to produce a more stable dataset during the model development phase by making modifications on a signal basis. After improving our model to an acceptable level of accuracy, our ultimate goal was to simplify the model to a level that would require less workload. The LibROSA library [1] was used to characterize sound features. To increase the robustness and generalization ability of the model across different music genres, data augmentation strategies using Gaussian noise and low-pass filters were used. Focusing on the performance of the models, we tried to demonstrate their effectiveness in predicting emotional states such as happiness, sadness, anger and relaxation in music files in our dataset. With the data augmentation strategies we used, we managed to significantly increase the model performance in terms of both accuracy and efficiency. In addition, we observed that the incompatibility problems that can be encountered during the processing of different audio files were completely eliminated. In summary, we believe that this study not only provides many technical contributions to the field of music emotion recognition,

but also provides outputs that can support future research at the intersection of technology, psychology and musicology.

Keywords: Music, emotion recognition, Deep learning, Signal Processing

ÖZET

DERİN SİNİR AĞLARI KULLANILARAK MÜZİK DUYGU TANIMLAMASI

PÜRE, HAKAN
Bilgisayar Mühendisliği Yüksek Lisans

Danışman: Assist. Prof. Dr. Abdül Kadir GÖRÜR
Mayıs 2024, 55 Sayfa

Müzik günümüzde milyarlarca dolarlık ekonomik değer taşımaktadır. Müziğin hissettirdiği duygunun doğru şekilde tespit edilmesi için doğru sınıflandırma yapılması gerekmektedir. Bu çalışmada müzikten duygu tespitinin ve sınıflandırmasının doğruluğunu artırmak amacıyla derin sinir ağlarını kullanarak müzik duygu tanıma (MER) konusunda kapsamlı bir araştırma sunmaya çalışmaktayız. Farklı ülkelerin müzikleri üzerine birçok çalışma yapılmış olmasına rağmen, Türk müziği üzerine çok az çalışma bulunmaktadır. Bu nedenle çalışmamızı, Türkçe şarkılardan oluşan bir veri seti kullanarak geliştirdik. Araştırmamızda, çeşitli ses özelliklerinin, melodi, armoni, ritim, karmaşık desenleri ile bu özelliklerin tetiklediği duygular arasındaki ilişkileri keşfetmek için derin öğrenme mimarilerini (CNN, LSTM) ve makine öğrenme algoritmasını (RFC) kullandık. Çalışmamızdaki ilk hedefimiz sinyal bazında modifikasyonlar yaparak model geliştirme aşamasında daha kararlı bir veri seti üretmekti. Modelimizi kabul edilebilir bir doğruluk seviyesine getirdikten sonra modeli daha az iş yükü gerektirecek seviyede basitleştirmek ise nihai hedefimizdi. Ses özelliklerinin karakterize edilmesini sağlamak amacıyla LibROSA kütüphanesi [1] kullanılmıştır. Modelin farklı müzik türleri arasında dayanıklılığını ve genelleme yeteneğini artırmak için gaussian gürültü ekleme ve alçak geçirgen filtrelerin uygulandığı veri artırma stratejileri kullanılmıştır. Modellerin performansına odaklanarak veri setimiz içindeki müzik dosyalarının mutluluk, üzüntü, öfke ve rahatlama gibi duygusal durumlarını tahmin edilmesindeki etkinliklerini göstermeye çalıştık. Kullandığımız veri artırma stratejileriyle model performansını hem doğruluk hem de verimlilik açısından önemli ölçüde artırmayı başardık. Buna ek olarak farklı ses dosyalarının işlenmesi sürecinde karşılaşılabilen uyumsuzluk problemlerinin de

tamamen ortadan kalktığını gözlemledik. Özetle, bu çalışma sadece müzik duygu tanıma alanına birçok teknik katkı sağlamakla kalmayıp, aynı zamanda teknoloji, psikoloji ve müzikoloji alanlarının kesişim noktasında gelecekteki araştırmalara destek olabilecek çıktılar elde ettiğimizi düşünüyoruz.

Anahtar Kelimeler: Müzik duygu algılama, Derin öğrenme, Sinyal işleme

ACKNOWLEDGEMENT

First and foremost, I would like to express my deepest gratitude to my advisor, Assist. Prof. Dr. Abdül Kadir GÖRÜR, for his invaluable guidance, unwavering support, and insightful feedback throughout the course of this research. His expertise and encouragement have been instrumental in shaping this work, and I am truly grateful for his mentorship.

I would also like to extend my sincere thanks to the faculty members of the Department of Computer Engineering at Çankaya University for their knowledge and support during my studies. Their teachings and advice have significantly contributed to my academic and personal growth.

A special thank you goes to my wife, Yıldız PÜRE, and my twin daughters, Damla PÜRE and Yağmur PÜRE, whose unconditional love, patience, and encouragement have been my constant source of strength. Their belief in me has been a cornerstone throughout this challenging yet fulfilling endeavor.

This thesis is the culmination of not just my efforts but the collective contributions of all those who have supported me along the way. For that, I am deeply grateful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DOI	: Digital Object Identifier
Ed./Eds.	: Editors
Sect.	: Section
Ed.	: Editor
Art.	: Article
p./pp.	: Page / Pages
No.	: Number
URL	: Uniform Resource Locator
Vol.	: Volume
Y.	: Year
MER	: Music Emotion Recognition
PCM	: Pulse Code Modulation
MPEG	: Moving Picture Experts Group
LSTM	: Long Short-Term Memory
MFCC	: Mel Frequency Cepstral Coefficients
CNN	: Convolutional Neural Network
RFC	: Random Forest Classifier
ReLU	: Rectified Linear Unit
Kb/s	: Kilobits Per Second
kHz	: Kilohertz
Et al.	: Latin Term *et alii* meaning “And Others”

CHAPTER I
INTRODUCTION AND PREPARATIONS

1.1 INTRODUCTION

Music, with a history dating back to ancient times, is one of the art forms people use to express their emotions. The classification and modeling of these elements will aim to determine which emotion the tested music evokes. The classification task will be conducted based on which of the four different regions the music falls into, to predict whether the tested music evokes an Angry, Happy, Sad, or Relax emotion (Figure 1).

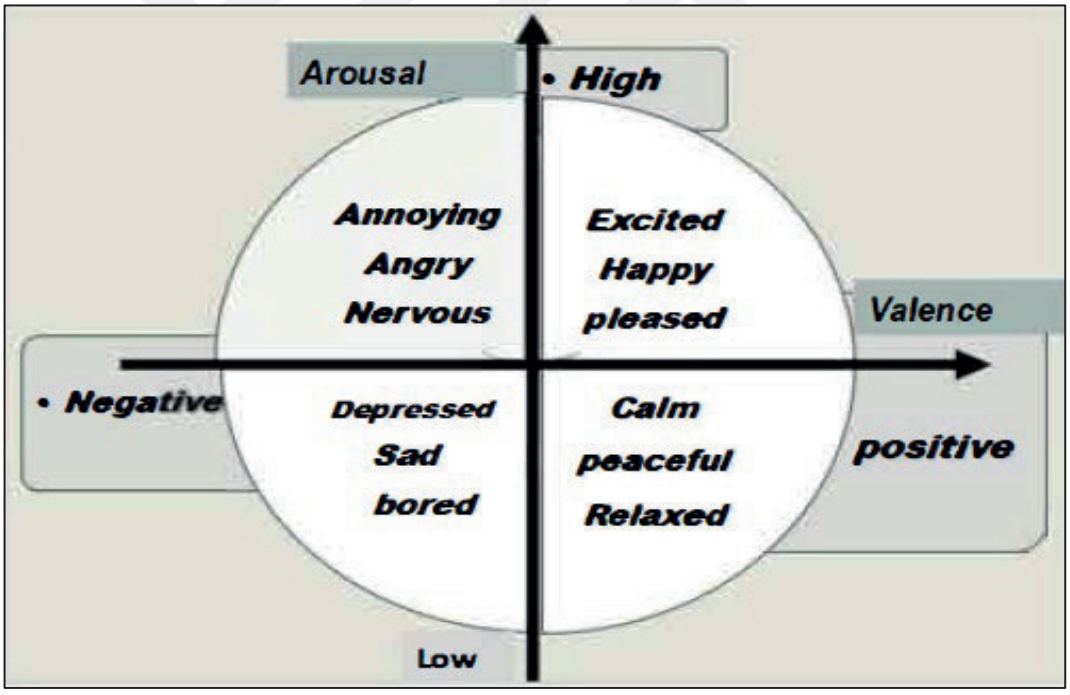

Figure 1: Russell (1980) proposed a two-dimensional emotion model [2]

Music has many elements, including loudness, pitch, beat, tempo, rhythm, melody, harmony, texture style, timbre, dynamics, structure, and more. This study has been prepared with the aim of developing a model that can operate with the highest possible accuracy and the lowest possible cost.

1.2 PREVIOUS STUDIES AND RESULTS

We conducted a literature review to learn about the methods used in previous studies in this field and to gather information on the performance rates achieved. We first examined studies conducted on datasets consisting of songs produced in Turkey, followed by research utilizing datasets composed of songs from foreign countries. As a result of this review, we compiled a table listing the accuracy rates achieved and the methods used in these studies (Table 1).

Table 1: Previous Studies And Results

Authors	**Features**	**Models**	**Accuracy**
[3]	Concatenated Features	ANN	75%
[4]	Matlab Audio Analysis Library	Logistic Regression	48%
[5]	LibROSA HSF	RNN (LSTM)	82%
[6]	MFCC	1D CNN + Bidirectional Long Short-Term (BiLSTM)	94%
[7]	Chromagram	2D CNN	80,6%
[8]	Concatenated Features	Bi-Modal Deep Boltzman Machine	78,5%
[9]	Concatenated Features	ANN	79,3%
[10]	Concatenated Features	Fuzz K-Nearest Neighbors (FKNN)	82,7%

1.3 SELECTING OF THE DATASET AND DATA PREPROCESSING

Although there have been many studies on music emotion recognition (MER), research that uses Turkish music as a dataset is extremely limited. In this study, a dataset composed of Turkish-language songs, the Turkish Music Emotion Dataset [3], was used. The selected dataset includes songs rich in various musical genres and emotional content, enhancing the model's generalization ability. The dataset was meticulously selected and labeled to cover a wide emotional spectrum in Turkish music. The songs were categorized into common emotional categories such as happiness, sadness, anger, and relaxation. There are 400 songs in total, each 30 seconds long, 100 songs expressing these four emotional states. The original versions of the files were in MPEG Audio version 1 Layer 3 format with a variable bit rate of 192 Kb/s and a sampling rate of 44.1 kHz. In our experiments, we experienced many problems due to incompatibility while processing MPEG Audio (.mp3) files to extract features. To overcome these problems and prevent possible data loss due to compression in the original files, we first converted all sample files to PCM (.wav) format with a constant bit rate of 1,411 Kb/s and a sampling rate of 44.1 kHz. While performing these conversions, we applied an 18.6 kHz low-pass filter to the files while converting from the original MP3 files to PCM files. The human ear does not perceive frequencies above 20 kHz. Therefore, we aimed to eliminate temporary noise on the signal and increase feature extraction performance with the filter we applied. This filtering resulted in our data being limited to approximately 16.5 kHz. This conversion had two main purposes: first, to eliminate compatibility issues and second, to prevent potential data loss due to compression in the original files. As a result of this conversion, file sizes increased by approximately nine times. After this change, the new dataset was processed using the librosa library to extract the audio features of each song. In addition to fundamental musical elements such as melody, harmony, rhythm, spectral, temporal, and rhythmic features were also extracted and transformed into inputs for the model. These features play a critical role in accurately predicting emotional states during the model's learning process.

CHAPTER II
VISUALISING THE TEST DATASET

2.1 SPECTROGRAM AND SPECTRUM ANALYSIS OF THE TEST FILES

Our dataset was divided into classes to represent four different emotions. During our literature review, we found that nearly all studies began by analyzing audio files to ensure correct steps were followed in the feature extraction process. Similarly, we performed spectrum and spectrogram analyses on our dataset. These analyses allowed us to visualize the frequency levels, amplitudes, intensities, and variations produced by songs in our dataset that conveyed similar emotional expressions. Our dataset contained a total of 400 song files, with 100 files representing each of the four emotional states, and we repeated this test for each file. The results of these analyses revealed that music files with similar emotional expressions exhibited similar patterns. The selected sample graphs for each emotional state are as follows: Angry (Figure 2), Happy (Figure 3), Relaxed (Figure 4), Sad (Figure 5). When we analyzed the music files played instrumentally across all four of our classification groups, we observed that the concentration occurred at lower decibel levels. However, the frequency, intensity, and amplitude characteristics still showed similar patterns for instrumental songs in each emotional state (Figure 6).

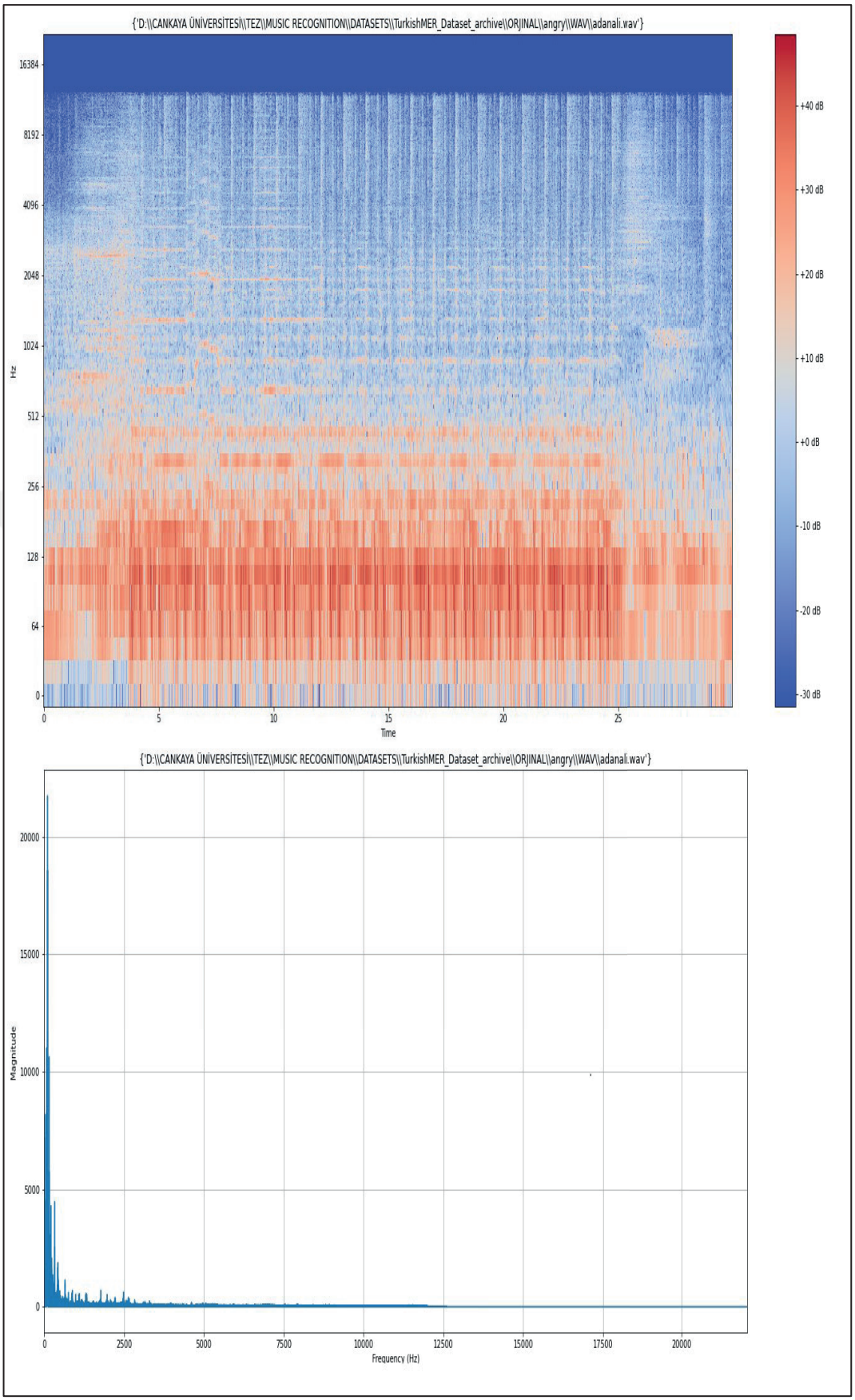

Figure 2: Spectrogram and Spectrum Analysis of Angry Audio Sample

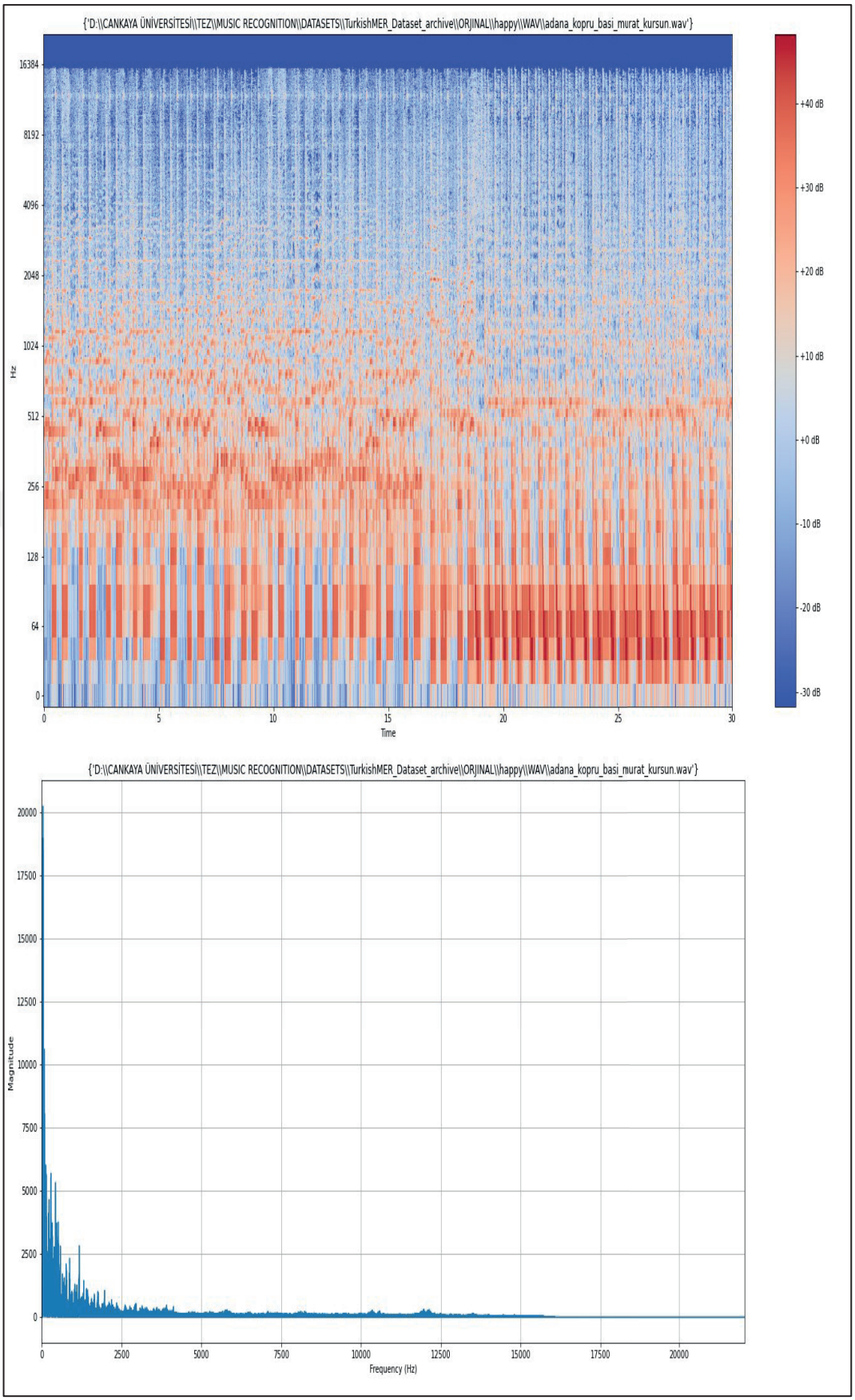

Figure 3: Spectrogram and Spectrum Analysis of Happy Audio Sample

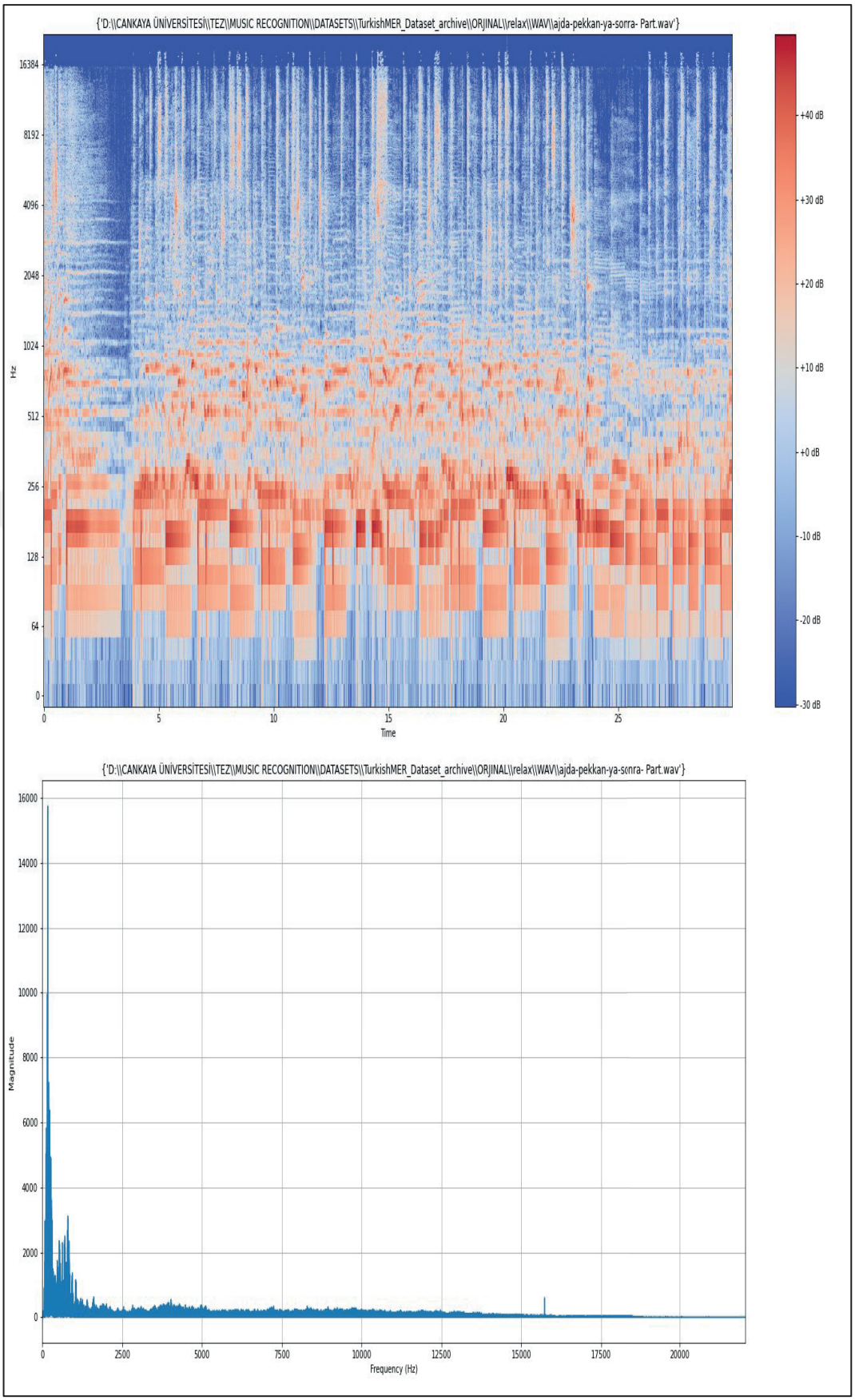

Figure 4: Spectrogram and Spectrum Analysis of Relax Audio Sample

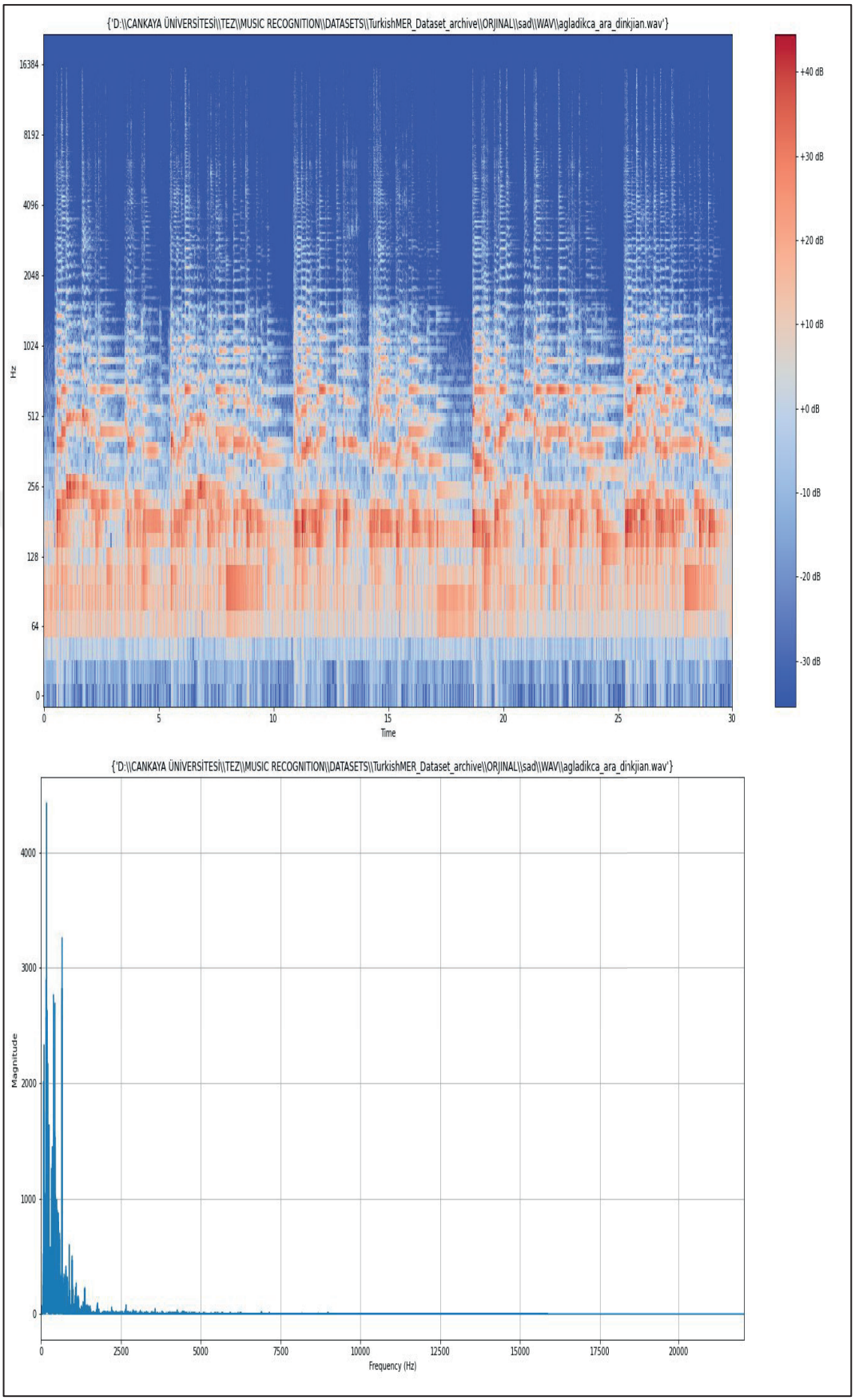

Figure 5: Spectrogram and Spectrum Analysis of Sad Audio Sample

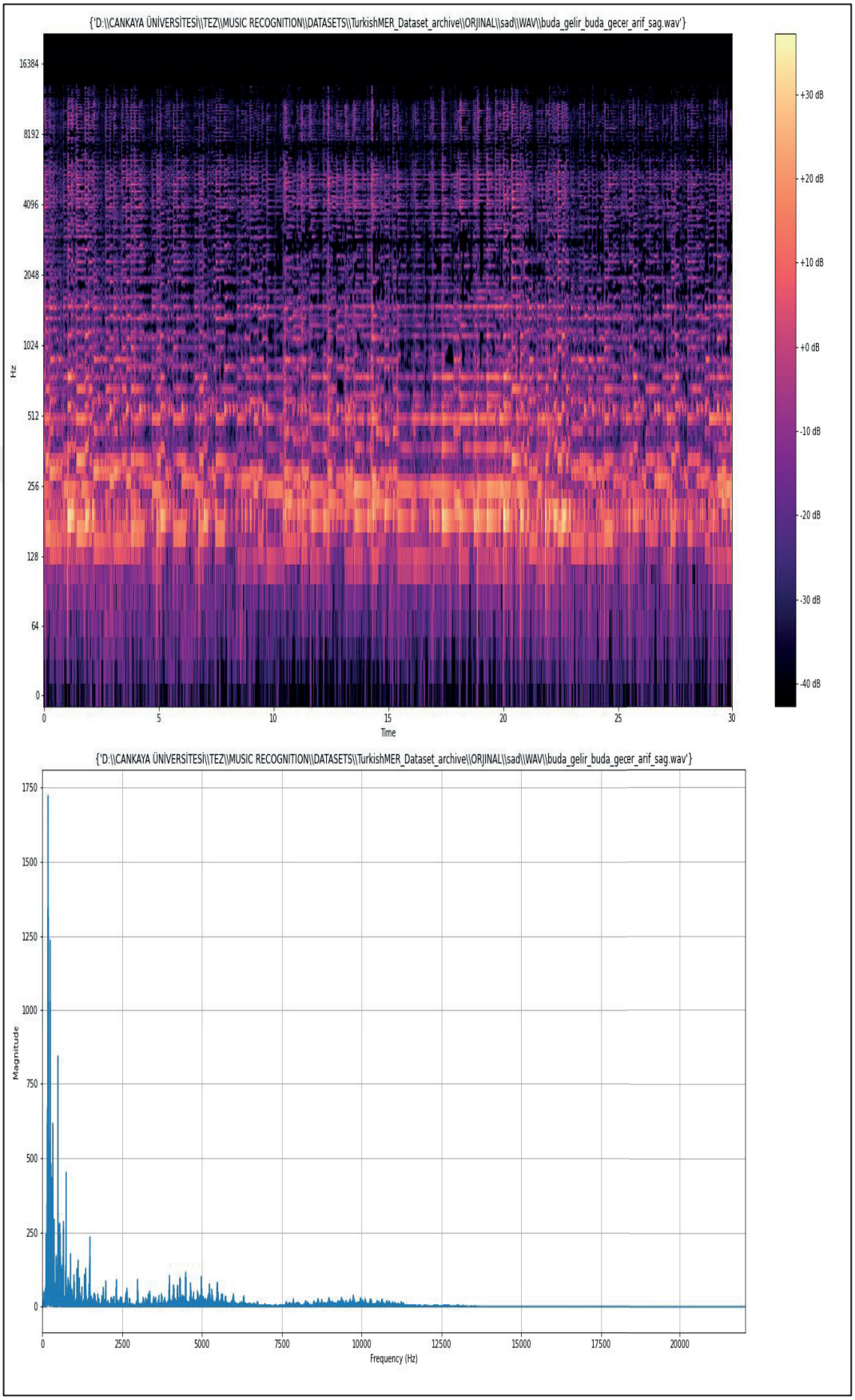

Figure 6: Spectrogram and Spectrum Analysis of Sad Instrumental Audio Sample

CHAPTER III
MODELLING PHASE

3.1 COMBINATION OF CNN WITH LSTM NETWORKS

The structure of the model is a combination of Convolutional Neural Network (CNN) and Long Short-Term Memory LSTM based neural network. The flowchart of this model presented as (Figure 7).

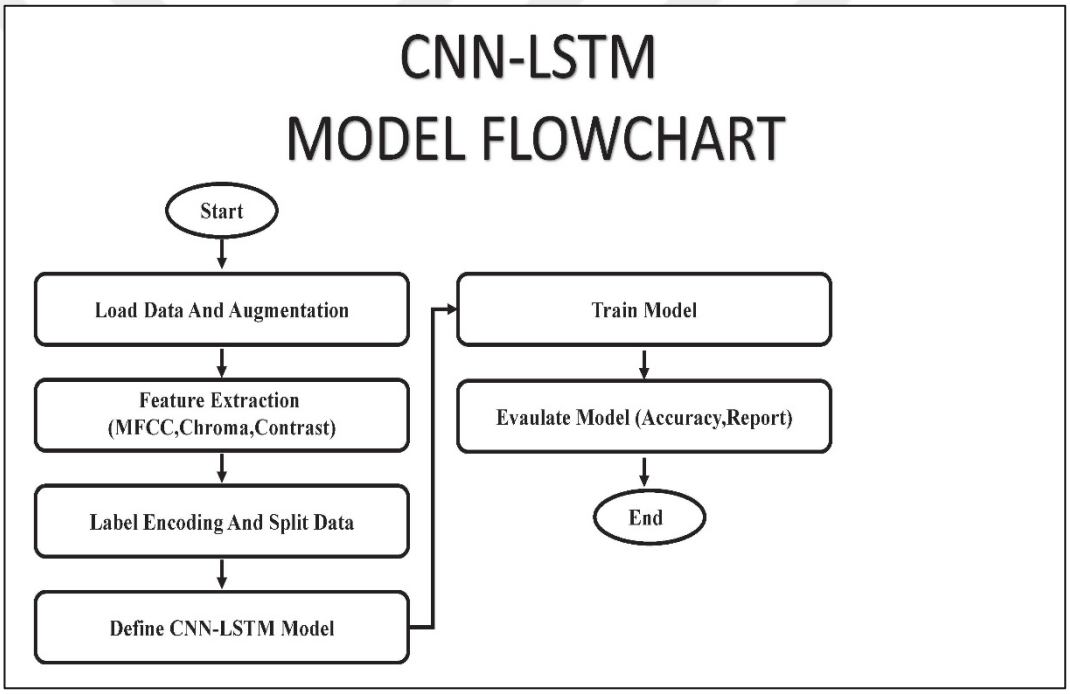

Figure 7: CNN-LSTM Combination Model Flowchart Diagram

This model processes the audio files by extracting MFCC features, encoding emotions, splitting the data into training and test sets, and then training a model to classify the emotions. The labels are encoded using LabelEncoder and converted to one-hot encoding using to_categorical. The dataset is split into training and testing sets using train_test_split.The model training was conducted using 80% of the songs in the dataset for training and 20% for validation. The model is a sequential neural network

composed of two convolutional layers followed by batch normalization and maxpooling, two LSTM layers, with dropout for regularization and a fully connected layer and an output layer using softmax activation for multi-class classification. The model is compiled with the Adam optimizer and categorical cross-entropy loss.

3.1.1 Hyperparameters Of CNN-LSTM Combination Model

We use following parameters for data splitting hyperparameters of our first model.

test_size=0.2 parameter for data splitting. 20% of the dataset is reserved for testing, while the remaining 80% is used for training.

random_state=42 ensures reproducibility. We use the "audiomentations" library which is a powerful techique to make the model more by providing it with varied versions of input data.

We applies Gaussian noise, time stretching, and pitch shifting to the audio data to increase the diversity of the training dataset. To reduce the sensitivity of our model to small changes in the test files and to increase its robustness and correct prediction rate, we added random noise to the signal before processing the file, formula (3.1).

$$y\_noise = y + noise\_factor * noise \tag{3.1}$$

By incorporating noise addition and various data augmentation techniques into our modeling process, we anticipate that the model will be better able to generalize to unseen data. Along with the noise addition process, we also used pitch shifting and time stretching techniques to enrich the data we obtained. In this way, we extracted various features from the modified audio clips (including MFCCs, chroma, mel spectrogram, spectral contrast, tonnetz, spectral rolloff, and centeroid) and combined them to create a final feature vector. Our main expectation while making these changes was to increase the robustness of our model by simulating real-world scenarios. We thought that by including noise addition and various data augmentation techniques in our modeling process, the model would be better able to generalize to unseen data that it was not tested on during training.

min_amplitude=0.001, max_amplitude=0.015: These parameters control the amplitude range of the Gaussian noise added to the audio signal. Higher values can

make the model more robust to noisy data, but if set too high can also lead to model learning noise instead of useful patterns.

min_rate=0.8, max_rate=1.25: These parameters define the range for stretching the time axis of the audio signal. It effectively simulates variations in playback speed. A broad range makes the model more flexible but can distort the audio too much if overdone.

min_semitones=-4, max_semitones=4: These parameters control the pitch shifting applied to the audio signal. It changes the pitch by a number of semitones. A wider range introduces more variety but might alter the musical quality significantly and making it harder for the model to learn.

Conv1D(64, kernel_size=3, activation='relu'):

64: The number of filters determines how many features the convolutional layer will learn. More filters generally capture more patterns, but they increase computational complexity.

kernel_size=3: This determines the size of the filter. A smaller kernel size captures fine-grained details, while a larger one captures broader patterns. A kernel size of 3 is typical for detecting short-term dependencies in audio data.

activation='relu': ReLU (Rectified Linear Unit) activation introduces non-linearity, which helps the model learn complex patterns.

BatchNormalization():This normalizes the output of the previous layer, stabilizing the learning process and often speeding up training. It also helps to reduce internal covariate shift, improving model generalization.

MaxPooling1D(2): This reduces the dimensionality of the output, effectively down-sampling the feature map by taking the maximum value over a window of size 2. Pooling helps in reducing computation and controlling overfitting.

Conv1D(128, kernel_size=3, activation='relu'):

128: This layer has more filters than the first convolutional layer, allowing it to learn more complex and abstract features.

kernel_size=3: The same kernel size as the first layer, ensuring consistency in feature extraction at this stage.

activation='relu': ReLU (Rectified Linear Unit) activation introduces non-linearity, which helps the model learn complex patterns.

MaxPooling1D(2): Similar to the first pooling layer, this further reduces the dimensionality, leading to a more abstract representation of the input data.

LSTM(64, return_sequences=True):

64: This specifies the number of units in the LSTM cell. The more units, the more capacity the LSTM has to learn temporal dependencies, but this also increases the risk of overfitting and the computational load.

return_sequences=True: This ensures that the LSTM outputs the full sequence of outputs (one for each time step), which is necessary when stacking LSTM layers.

Dropout(0.5): This is a regularization technique that randomly drops 50% of the units in the layer during training. This helps to prevent overfitting by ensuring the model doesn't become too reliant on any single neuron.

LSTM(32): 32: The second LSTM layer has fewer units, which helps in gradually reducing the dimensionality of the data as it flows through the network, focusing on the most important temporal patterns.

Dropout(0.5): Dropout is applied again to reduce overfitting.

Dense(32, activation='relu'): 32: This fully connected layer has 32 units, which condenses the information extracted by the previous layers.

activation='relu': ReLU is used again to introduce non-linearity.

Dense(len(data_paths), activation='softmax'):

len(data_paths): This output layer has as many units as there are emotion categories (angry, happy, relax, sad).

activation='softmax': With a softmax activation function to output probabilities for each category.

epochs=50: The number of epochs determines how many times the model sees the entire dataset during training. More epochs allow the model to learn better but increase the risk of overfitting if the model continues to improve on training data but not on validation data.

batch_size=32: The batch size defines how many samples are processed before the model’s internal parameters are updated. Smaller batch sizes lead to more updates and can potentially lead to a more finely tuned model, but they also increase computational time. All hyperparameters shown in (Figure 8).

Hyperparameter	Description	Possible Values	Real Values	Effect on Model
LSTM Units	Number of units (neurons) in the LSTM layers.	32, 64, 128, 256	64, 32	More units capture complex patterns but may lead to overfitting and increased training time.
Dropout Rate	Fraction of input units to drop to prevent overfitting.	0.2, 0.3, 0.5	0.5, 0.5	Higher dropout prevents overfitting but may cause underfitting if too high.
Conv1D Filters	Number of filters in the Conv1D layers.	32, 64, 128	64, 128	More filters capture more features but increase computational cost and overfitting risk.
Kernel Size (Conv1D)	Size of the convolutional kernel in Conv1D layers.	3, 5, 7	3, 3	Smaller kernels capture finer patterns, larger kernels capture broader patterns.
Dense Units	Number of neurons in the Dense layers after the LSTM.	16, 32, 64	32	More units increase model capacity but may cause overfitting.
Batch Size	Number of samples per gradient update during training.	16, 32, 64, 128	32	Larger batch sizes speed up training but may lead to suboptimal convergence and require more memory.
Epochs	Number of complete passes through the training dataset.	10, 20, 50, 100	50	More epochs improve performance but can lead to overfitting.
Learning Rate	Step size for optimizing the model during training.	0.001, 0.0001, 0.01	0.001	Higher learning rates speed up training but can cause suboptimal convergence.
Optimizer	Algorithm used for minimizing the loss function.	Adam, SGD, RMSprop	Adam	Different optimizers impact training speed and final performance
Augmentation	Data augmentation techniques applied to input audio data.	Gaussian Noise, Pitch Shift, Time Stretch	Gaussian Noise, Pitch Shift, Time Stretch	Augmentation improves generalization by increasing training data diversity.

Figure 8: CNN-LSTM Combination Model Hyperparameters

3.1.2 Experiments And Results Of CNN-LSTM Combination Model

Techniques such as dropout and data augmentation were applied during training to prevent overfitting and improve the model's overall performance. The initial results indicate that the model achieved promising accuracy in predicting emotional states. The model performed particularly well in the angry and happy categories. However, it was observed that further improvements and fine-tuning are necessary to increase accuracy in the sad and relax categories. The model's accuracy reached 56%, shown in Table 2.

Table 2: Classification Report of The Model

	Precision	Recall	F1-Score	Support
Angry	0.71	0.58	0.64	26
Happy	1	0.39	0.56	18
Relax	0.52	0.72	0.60	18
Sad	0.37	0.56	0.44	18
Accuracy			0.56	80
Macro Avg.	0.65	0.56	0.56	80
Weighted Avg.	0.66	0.56	0.57	80
Accuracy	**0.5625**			

3.2 2D CNN NETWORKS

Previous results that we reached shows that deep neural network-based approaches have strong potential in the field of music emotion recognition. However, to further improve the model's performance, it will be necessary to experiment with different architectures. Therefore we changed our model to another architecture 2D CNN model. The flowchart of this model presented as (Figure 9).

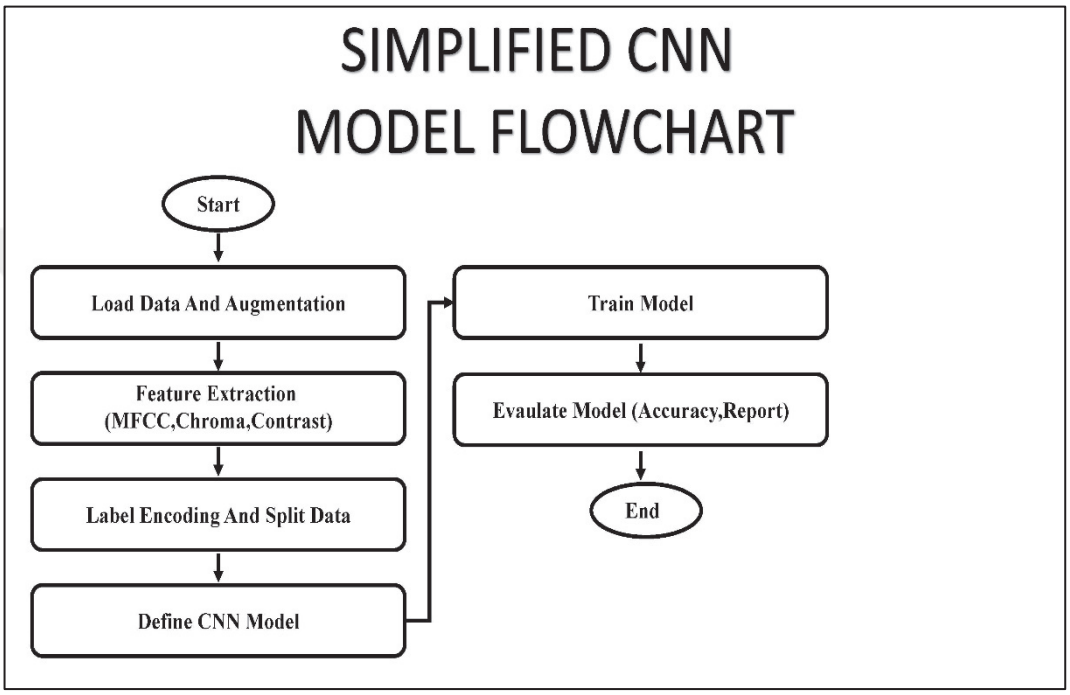

Figure 9: 2D CNN Model Flowchart Diagram

In our previous model we used audiomentations library for data augmentation, applying Gaussian noise, time-stretching, and pitch-shifting directly to the audio before extracting features. We extracted MFCCs, chroma features, and spectral contrast and then we combined these features into a single feature vector for each audio file. On the other hand, while we changed model to 2D CNN architecture we augmented data by applying pitch shifting, time stretching, and noise injection within the extract_features function. We extracted more features compared to the first code, including MFCCs, chroma, mel-spectrogram, spectral contrast, tonnetz, spectral rolloff, and spectral centroid. After that we combined all these features into a comprehensive feature vector and reshaped into 2D arrays before being fed into the

model, which applies 2D convolutions. Labels are encoded into integer form using LabelEncoder and then one-hot encoded using to categorical. We used StandardScaler for scaling the features before feeding them into the both models to have zero mean and unit variance. But this time we reshaped the features into 13x15x1 3D arrays to match the input shape required by the 2D CNN. We trained our model for 50 epochs previously. We used similar evaluation metrics in second model but we used 20 epochs for increasing performance during training.

3.2.1 Hyperparameters Of 2D CNN Model

20% of the dataset is reserved for testing, while the remaining 80% is used for training. Following augmentations and feature extraction techniques provided varied training data, reducing overfitting and enhancing the model's ability to generalize across different audio conditions.

noise_factor=0.008: For injecting Gaussian noise into the audio signal, with the noise level controlled by this factor.

n_steps=4: For shifting the pitch of the audio signal by 4 semitones.

rate=1.3. For stretching the audio signal's duration by a factor of 1.3.

Following comprehensive feature set captures the timbral, harmonic, and rhythmic characteristics of the audio, which are critical for differentiating between emotions.

MFCCs: n_mfcc=40: For extracting 40 Mel-frequency cepstral coefficients, which are essential features for analyzing the timbre of audio.

Chroma: Extracted using librosa.feature.chroma_stft

Mel Spectrogram: Extracted using librosa.feature.melspectrogram

Spectral Contrast: Extracted using librosa.feature.spectral_contrast

Tonnetz: Extracted using librosa.feature.tonnetz

Spectral Rolloff: Extracted using librosa.feature.spectral_rolloff

Spectral Centroid: Extracted using librosa.feature.spectral_centroid

Following steps are necessary for ensuring the labels are in the correct format for training the CNN model using categorical cross-entropy loss.

Label Encoder: Converts string labels into integers.

One-Hot Encoding: Converts the integer labels into categorical format for multiclass classification.

Following combination of convolutional layers for feature extraction, followed by dense layers for classification, is a standard and effective CNN architecture for music or image-like data. The use of dropout layers mitigates overfitting.

Input(shape=input_shape): Specifies the input shape as (13, 15, 1), where 13x15 corresponds to the reshaped feature dimensions and 1 is the channel dimension.

Conv2D(32, (3, 3), activation='relu', padding='same') Uses 32 filters, each of size 3x3, with ReLU activation. The padding='same' ensures the output size remains the same as the input size.

MaxPooling2D(2, 2): Reduces the spatial dimensions by a factor of 2.

Dropout(0.25) and Dropout(0.5): Adds dropout with probabilities 0.25 and 0.5 to prevent overfitting by randomly dropping units during training.

Flatten Layer: Converts the 2D feature maps into a 1D vector.

Dense(128, activation='relu'): Fully connected layer with 128 units.

Dense(num_classes, activation='softmax'): Output layer with units equal to the number of classes, using softmax activation for multi-class classification.

Optimizer: adam : The Adam optimizer is widely used due to its adaptive learning rate and momentum features.

categorical_crossentropy: The categorical cross-entropy loss function is suitable for multi-class classification tasks.

Epochs: 20: The number of epochs determines how many times the model sees the entire dataset during training. More epochs allow the model to learn better but increase the risk of overfitting if the model continues to improve on training data but not on validation data. All hyperparameters shown in (Figure 10).

Hyperparameter	Description	Possible Values	Real Values	Effect on Model
LSTM Units	Number of units (neurons) in the LSTM layers.	32, 64, 128, 256	64, 32	More units capture complex patterns but may lead to overfitting and increased training time.
Dropout Rate	Fraction of input units to drop to prevent overfitting.	0.2, 0.3, 0.5	0.5, 0.5	Higher dropout prevents overfitting but may cause underfitting if too high.
Conv1D Filters	Number of filters in the Conv1D layers.	32, 64, 128	64, 128	More filters capture more features but increase computational cost and overfitting risk.
Kernel Size (Conv1D)	Size of the convolutional kernel in Conv1D layers.	3, 5, 7	3, 3	Smaller kernels capture finer patterns, larger kernels capture broader patterns.
Dense Units	Number of neurons in the Dense layers after the LSTM.	16, 32, 64	32	More units increase model capacity but may cause overfitting.
Batch Size	Number of samples per gradient update during training.	16, 32, 64, 128	32	Larger batch sizes speed up training but may lead to suboptimal convergence and require more memory.
Epochs	Number of complete passes through the training dataset.	10, 20, 50, 100	50	More epochs improve performance but can lead to overfitting.
Learning Rate	Step size for optimizing the model during training.	0.001, 0.0001, 0.01	0.001	Higher learning rates speed up training but can cause suboptimal convergence.
Optimizer	Algorithm used for minimizing the loss function.	Adam, SGD, RMSprop	Adam	Different optimizers impact training speed and final performance
Augmentation	Data augmentation techniques applied to input audio data.	Gaussian Noise, Pitch Shift, Time Stretch	Gaussian Noise, Pitch Shift, Time Stretch	Augmentation improves generalization by increasing training data diversity.

Figure 10: 2D CNN-LSTM Model Hyperparameters

3.2.2 Experiments And Results Of 2D CNN Model

Our new model 2D CNN with new modified hyperparameters in this part are designed to build a robust model for emotion recognition from music data. The augmentations ensure that the model generalizes well, while the CNN layers effectively capture the spatial patterns in the extracted features. This new model's accuracy reached 68%, shown in Table 3.

Table 3: Classification Report of the 2D CNN Model

	Precision	Recall	F1-Score	Support
Angry	0.71	0.65	0.68	26
Happy	0.73	0.89	0.80	18
Relax	0.70	0.78	0.74	18
Sad	0.57	0.44	0.50	18
Accuracy			0.69	80
Macro Avg.	0.68	0.69	0.68	80
Weighted Avg.	0.68	0.69	0.68	80
Accuracy	**0.6875**			

3.3 RANDOM FOREST CLASSIFIER MODEL

Despite all the optimization adjustments made to the previous architecture, there was no significant increase in accuracy. Therefore, we decided to replace the 2D CNN architecture with a Random Forest Classifier architecture. The primary reason for changing to an RFC (Random Forest Classifier) model at this point was the small size of our dataset. By changing to RFC, we aimed to reduce computational costs while maintaining performance, especially given the limited data available. Also, RFC models are particularly effective at capturing local patterns in melodies, rhythm, and harmonic structures, which allows for better identification of local relationships within the data. This capability makes RFC models more suitable for extracting the emotional characteristics of specific frequency bands or time segments in music, leading us to make this architectural change. Features are extracted similarly using MFCCs, chroma, mel-spectrogram, and others, but processed through a RFC instead of a deep learning model. And also, we employ GridSearchCV to optimize hyperparameters of the Random Forest model, which is a robust method for tuning. The flowchart of the model presented as (Figure 11).

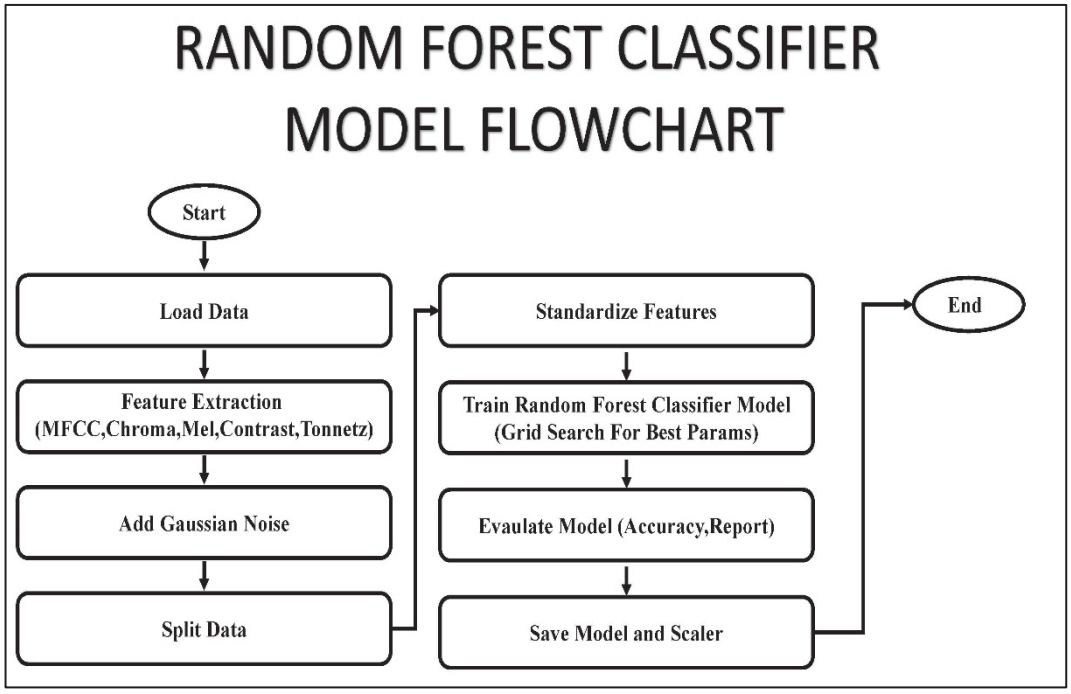

Figure 11: RFC Model Flowchart Diagram

With RFC model we applied a thorough process of feature extraction, data augmentation, and hyperparameter tuning to build a robust Random Forest model for music emotion recognition. By using GridSearchCV it is possible to tune multiple hyperparameters helps find the best-performing model. The chosen hyperparameters, especially those related to feature extraction and Random Forest configuration, are well-suited for this task, balancing accuracy with model complexity.

3.3.1 Hyperparameters Of Random Forest Classifier Model

test_size=0.2: 20% of the dataset is reserved for testing, while the remaining 80% is used for training.

random_state=42: Ensures reproducibility.

StandardScaler(): Standardizes the features by removing the mean and scaling to unit variance. Feature scaling ensures that all features contribute equally to the model's predictions, preventing features with larger ranges from dominating the learning process. RFC is distance-based model that is why this is important for our model.

n_estimators: The number of trees in the forest. More trees generally lead to better model performance because they reduce variance (the model becomes less sensitive to the specifics of any single training set). However, increasing the number of trees also increases computational cost and memory usage.

max_features='sqrt': The number of features to consider when looking for the best split. sqrt means the square root of the total number of features. Limiting the number of features considered at each split helps to prevent overfitting by introducing randomness and encouraging diversity among the trees. This is used for better generalization.

max_depth= [10, 20, None]: The maximum depth of the tree. Limiting the depth of the trees (e.g., to 10 or 20) prevents them from becoming too complex and overfitting to the training data. Allowing unlimited depth (None) can capture more complex patterns but may also lead to overfitting.

min_samples_split= [2, 5]: The minimum number of samples required to split an internal node.Higher values for min_samples_split prevent the model from learning overly specific patterns, reducing overfitting. However, setting this value too high might limit the model's ability to capture important patterns in the data.

min_samples_leaf= [1, 2]: The minimum number of samples required to be at a leaf node. Increasing min_samples_leaf ensures that leaf nodes have more than one sample, reducing the likelihood of overfitting. However, if set too high, it can prevent the model from capturing finer details.

bootstrap= [True, False]: Whether bootstrap samples are used when building trees. Using bootstrap sampling (True) adds randomness to the model, which generally helps in reducing overfitting by making the model less sensitive to the specifics of any single training set. In some cases, using the entire dataset (False) may lead to better performance.

cv=3: The number of cross-validation folds used in the grid search. Using 3-fold cross-validation ensures that the model's performance is evaluated on different subsets of the data, providing a more robust estimate of its generalization ability. More folds would provide a better estimate but would also increase computational time.

verbose=2: Controls the verbosity of the output during the grid search. A higher verbosity level provides more detailed output during the grid search, which can be useful for monitoring progress.

n_jobs=-1: Specifies the number of jobs to run in parallel. Setting n_jobs=-1 uses all available processors, speeding up the grid search. This is particularly important when dealing with large datasets or complex models.

All hyperparameters shown in (Figure 12).

Hyperparameter	Description	Possible Values	Real Values	Effect on Model
n_estimators	Number of trees in the random forest.	100, 200	200	More trees generally result in better performance but require more computational resources.
max_features	Number of features to consider when looking for the best split.	sqrt	sqrt	Using 'sqrt' ensures that a subset of features is considered, reducing overfitting.
max_depth	Maximum depth of the tree.	10, 20, None	None	Deeper trees capture more complex patterns but may overfit the data.
min_samples_split	Minimum number of samples required to split an internal node.	2, 5	2	A lower value splits nodes more frequently, capturing finer patterns but increasing the risk of overfitting.
min_samples_leaf	Minimum number of samples required to be at a leaf node.	1, 2	1	A lower value allows smaller leaf nodes, capturing more detail but increasing overfitting risk.
bootstrap	Method of sampling data points (with or without replacement).	True, False	True	Using bootstrap (sampling with replacement) improves model stability.
Add Gaussian Noise	Whether to add Gaussian noise to the audio signal for data augmentation.	True, False	True	Adding noise helps prevent overfitting by augmenting the training data.
Duration of Audio	Duration of the audio sample used for feature extraction.	30 seconds	30 seconds	Determines the amount of audio data processed for feature extraction.

Figure 12: Random Forest Classifier Model Hyperparameters

3.3.2 Experiments And Results Of Random Forest Classifier Model

In this new model, the feature extraction process was made more comprehensive by using multiple audio features. We extract 40 Mel-frequency cepstral coefficients and calculates the mean across time, chroma feature, representing the pitch class distribution and also tonal centroid features, representing the harmonic content. Computes the Short-Time Fourier Transform of the audio, used for calculating other features, mel spectrogram, representing the energy distribution across the mel scale and spectral contrast, capturing the difference in amplitude between peaks and valleys in the sound spectrum and stacks all these features into a single feature vector. We used RFC model using these features, optimizing its performance through grid search and then the trained model evaluated is saved for future use. With the changes made at this stage, the model's accuracy has increased to 76%, shown in Table 4.

Table 4: Classification Report of RFC Model

	Precision	**Recall**	**F1-Score**	**Support**
Angry	1	0.65	0.79	26
Happy	0.82	0.78	0.80	18
Relax	0.72	0.72	0.72	18
Sad	0.61	0.94	0.74	18
Accuracy			0.76	80
Macro Avg.	0.79	0.77	0.76	80
Weighted Avg.	0.81	0.76	0.77	80
Accuracy	**0.7625**			

3.4 OPTIMIZED RANDOM FOREST CLASSIFIER MODEL

One of the goals we set at the beginning of this study was to develop a model that could, if it achieved a meaningful accuracy rate, be made to run efficiently on portable devices with low resource consumption and minimal operational load. Although we have managed to increase the accuracy rate to 76% so far, we realized that for an application running on a laptop equipped with an Intel i7 processor and 512MB memory with Nvidia Cuda support, some cost-reducing changes could be made without compromising the model's accuracy. Therefore, we made the necessary changes to re-model using RFC, leveraging the modified dataset that we had previously optimized for performance in our model. This approach was intended to reduce computational costs while maintaining or improving accuracy. RFC is a powerful and flexible machine learning algorithm, but to achieve maximum performance, certain hyperparameters must be carefully adjusted. Therefore, in this step we adjust hyperparameters. The flowchart of the model presented as (Figure 13).

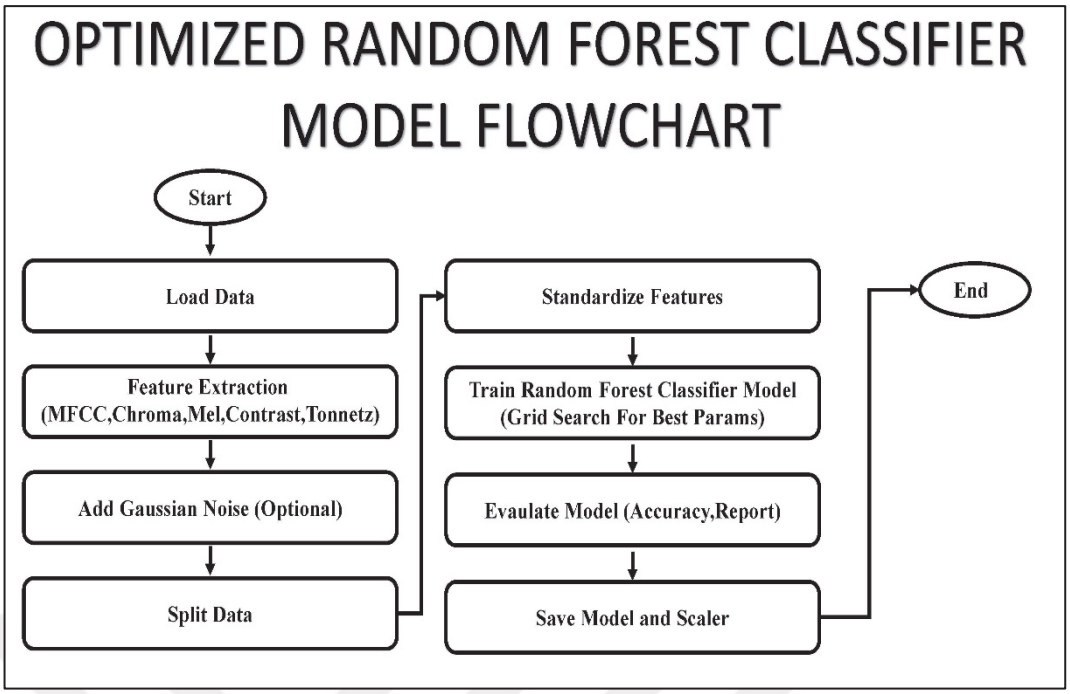

Figure 13: Optimized RFC Model Flowchart Diagram

In this latest version, we added an option to apply a low-pass filter during feature extraction, providing flexibility in the data preprocessing process. We made the filtering option customizable with parameters such as cutoff frequency and filter order. Using the LibROSA library, as in the previous version, we extracted features like MFCCs, chroma, mel-spectrogram, spectral contrast, and tonnetz. Additionally, we developed a more robust model with data augmentation using Gaussian noise. This addition allows for filtering in conjunction with noise injection, making the framework ready for experimenting with different data preprocessing techniques. We also used a similar Random Forest classifier, tuned via GridSearchCV for hyperparameter optimization.

3.4.1 Hyperparameters Of Optimized Random Forest Classifier Model

test_size=0.2: 20% of the dataset is reserved for testing, while the remaining 80% is used for training.

random_state=42 ensures reproducibility.

noise_factor=0.0001: This parameter controls the amplitude of the Gaussian noise added to the audio signal. A higher noise_factor results in more noise being added. Adding noise can help make the model more robust to variations in the input

data and prevent overfitting. However, if the noise level is too high, it could obscure the original signal, making it harder for the model to learn relevant patterns.

cutoff=5000: The cutoff frequency for the low-pass filter. Frequencies above this value are attenuated. A lower cutoff frequency removes more high-frequency content, which can help reduce noise but might also eliminate useful information. The choice of cutoff frequency depends on the characteristics of the audio signals and the desired features.

order=3: This parameter determines the steepness of the filter's transition band. A higher order filter has a steeper roll-off, meaning it separates more sharply between frequencies above and below the cut-off point. However, higher order filters can also introduce more phase distortion and computational complexity.

n_mfcc=40: This parameter determines the number of Mel-frequency cepstral coefficients (MFCC) extracted from the audio files. Extracting more MFCCs captures more detailed spectral information, which can help the model distinguish between different emotions in music. However, it also increases the dimensionality of the feature space, which may require more computational resources and can lead to overfitting if not properly regularized. Therefore, we choose 40 cepstral coefficients in our model.

n_estimators= [100, 200]: The number of trees in the Random Forest Classifier. More trees generally improve performance but increase computation time. A larger number of trees can reduce model variance, leading to better generalization. However, this comes with increased computational cost.

max_features=['sqrt']: The number of features to consider when looking for the best split. Using 'sqrt' selects the square root of the total number of features. Limiting the number of features considered at each split can prevent overfitting by introducing randomness and encouraging diversity among trees.

max_depth= [10, 20, None]: The maximum depth of the trees. Setting this to None allows the trees to grow until all leaves are pure or until they contain fewer than

min_samples_split samples. Limiting the depth of the trees prevents them from becoming too complex and overfitting the training data. However, deeper trees can capture more complex patterns.

min_samples_split= [2, 5]: The minimum number of samples required to split an internal node. Higher values prevent the model from learning overly specific patterns, which reduces overfitting but may also miss important details if set too high.

min_samples_leaf= [1, 2]: The minimum number of samples required to be at a leaf node. Increasing this value reduces the likelihood of overfitting by ensuring that each leaf has more than one sample, but it could limit the model's ability to capture fine details.

bootstrap= [True, False]: Whether bootstrap samples are used when building trees. Bootstrapping introduces randomness and can lead to a more robust model. Using bootstrapping helps to improve generalization by reducing overfitting. Not using it might result in a model that is too closely tied to the training data. All hyperparameters shown in (Figure 14).

Hyperparameter	Description	Possible Values	Real Values	Effect on Model
n_estimators	Number of trees in the random forest.	100, 200	200	More trees generally result in better performance but require more computational resources.
max_features	Number of features to consider when looking for the best split.	sqrt	sqrt	Using 'sqrt' ensures that a subset of features is considered, reducing overfitting.
max_depth	Maximum depth of the tree.	10, 20, None	None	Deeper trees capture more complex patterns but may overfit the data.
min_samples_split	Minimum number of samples required to split an internal node.	2, 5	2	A lower value splits nodes more frequently, capturing finer patterns but increasing the risk of overfitting.
min_samples_leaf	Minimum number of samples required to be at a leaf node.	1, 2	1	A lower value allows smaller leaf nodes, capturing more detail but increasing overfitting risk.
bootstrap	Method of sampling data points (with or without replacement).	True, False	True	Using bootstrap (sampling with replacement) improves model stability.
Add Gaussian Noise	Whether to add Gaussian noise to the audio signal for data augmentation.	True, False	True	Adding noise helps prevent overfitting by augmenting the training data.
Duration of Audio	Duration of the audio sample used for feature extraction.	30 seconds	30 seconds	Determines the amount of audio data processed for feature extraction.
Apply Low-Pass Filter	Whether to apply a low-pass filter to the audio signal.	True, False	False	Applying a filter may reduce noise but could also remove important features from the signal.

Figure 14: Optimized Random Forest Classifier Model Hyperparameters

3.4.2 Experiments And Results Of Optimized RFC Model

At this step by fine tuning hyper parameters new models are carefully designed to balance model complexity, generalization, and computational efficiency. The additional preprocessing steps, gaussian noise adding, low pass filtering provide robustness against noisy data, while the grid search ensures the optimal combination of parameters for the RFC. The feature extraction parameters allow the model to capture a rich set of audio characteristics, which are essential for accurate emotion recognition. With the changes made at this stage, the model's accuracy has increased to 81%, shown in Table 5.

Table 5: Classification Report of Optimized Random Forest Classifier Model

	Precision	Recall	F1-Score	Support
Angry	0.94	0.85	0.89	20
Happy	0.70	0.95	0.81	20
Relax	0.82	0.70	0.76	20
Sad	0.83	0.75	0.79	20
Accuracy			0.81	80
Macro Avg.	0.72	0.81	0.81	80
Weighted Avg.	0.73	0.81	0.81	80
Accuracy	**0.8125**			

3.4.3 Visualize Optimized RFC Model Performance

These visualization tools will help assess the performance and interpretability of the RFC model in the context of music emotion recognition. In order to visualize model performance, several functions are added:

Feature Importance graph visualizes the importance of different features as determined by the RandomForestClassifier (Figure 15). Confusion Matrix graph visualize the classification performance of the model (Figure 16). Receiver Operating Characteristic (ROC Curve) graphs visualizes for each class, showing the balance between true positive rate and false positive rate (Figure 17). Precision-Recall Curve graphs illustrate the balance between precision and recall (Figure 18).

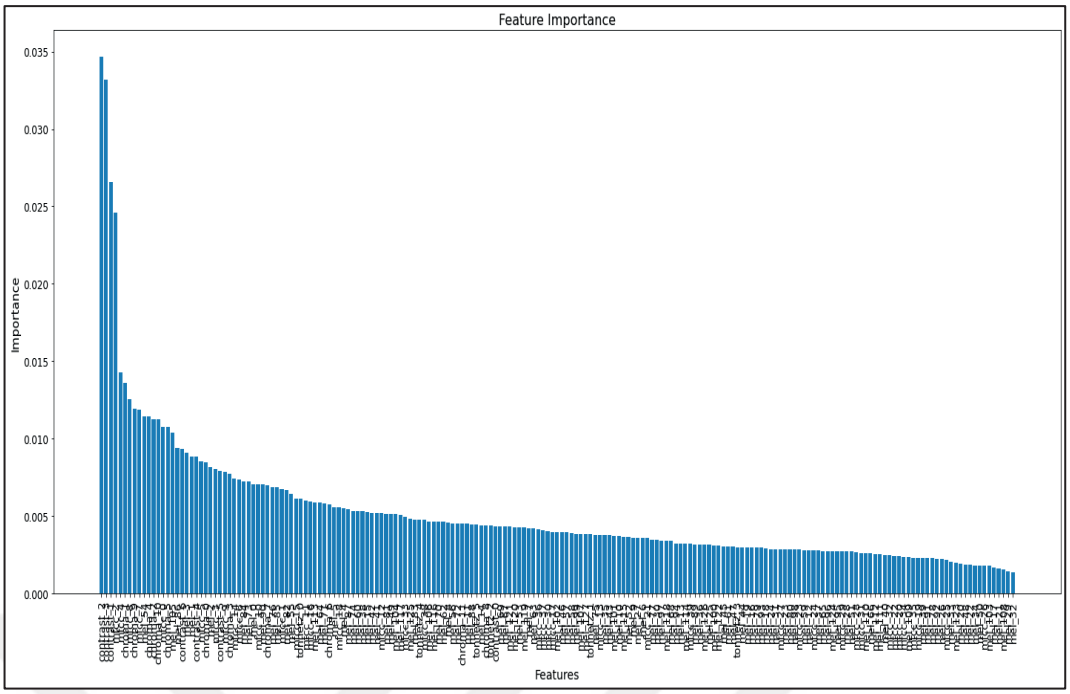

Figure 15: Plot Feature Importance of Optimized RFC Model

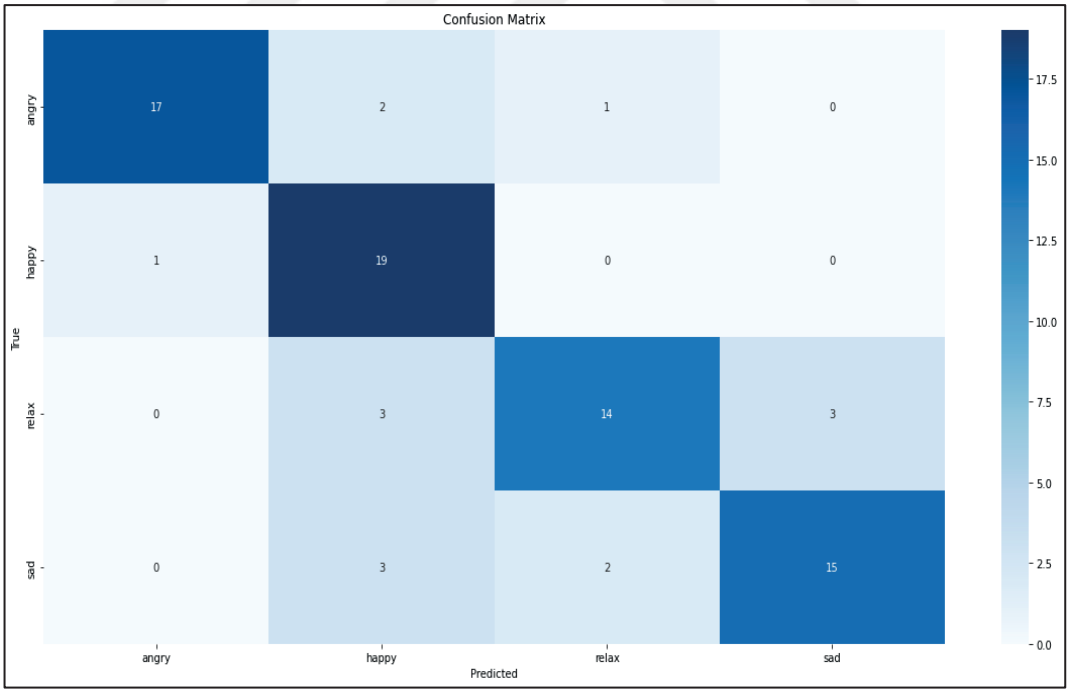

Figure 16: Plot Confusion Matrix of Optimized RFC Model

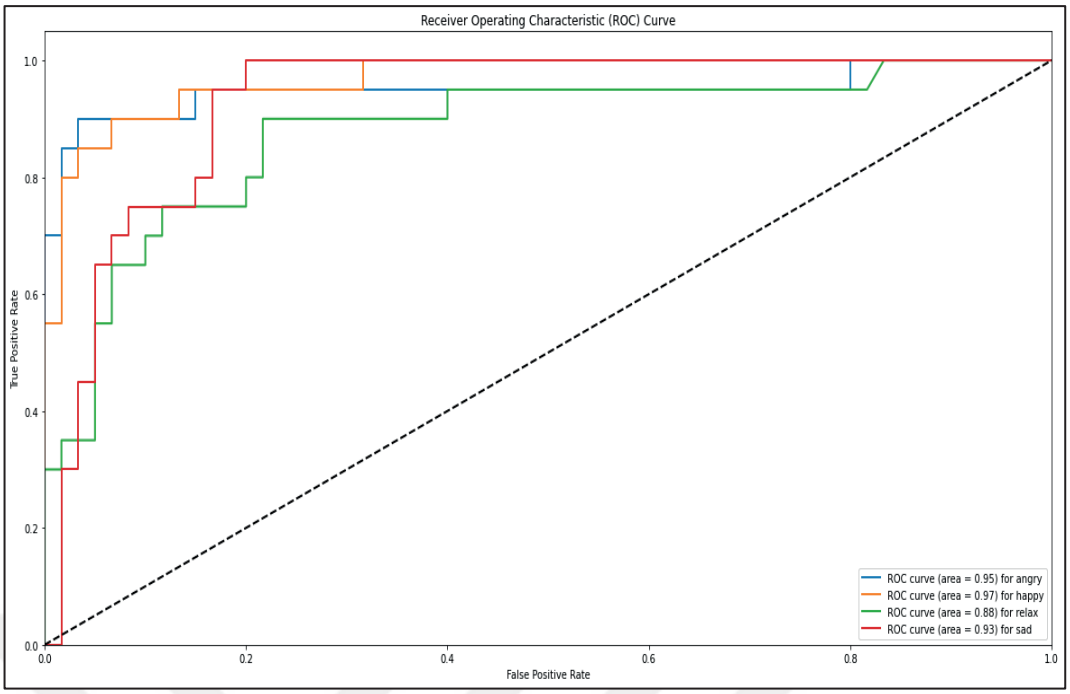

Figure 17: Plot ROC Curve of Optimized RFC Model

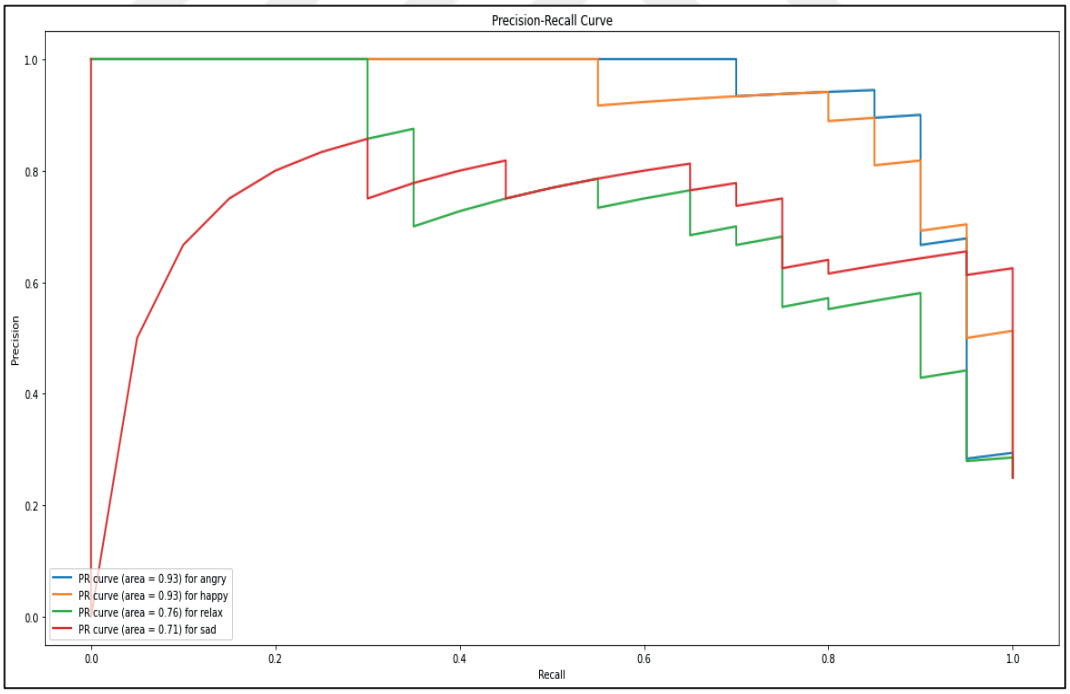

Figure 18: Plot Precision Recall Curve of Optimized RFC Model

CHAPTER IV
TESTING PHASE

4.1 TEST MODEL

Throughout our study, we used a dataset consisting of a total of 400 songs, evenly distributed with 100 songs representing each of the four emotional states. During the training of our final optimized RFC model, 80% of our dataset was used for training, and the remaining 20% was used to test the trained model. At this stage, our model achieved an accuracy rate of 81%. To further test the stability of our model on the entire dataset and determine the final accuracy rate, we developed a code to test the RFC model we built and saved on all 400 songs. In this phase, we aimed to measure both the accuracy and performance of our model. The test results showed that each prediction took approximately 0.01 seconds and accurately predicted the emotional states of the songs with an accuracy rate of 89.75%. Based on these results, we demonstrated that our model operates consistently (Table 6).

Table 6: Performance Analyze Of Optimized RFC Model Over Turkish Dataset

Real Emotion	Happy Prediction	Sad Prediction	Angry Prediction	Relax Prediction	Acc. %	Avarage Prediction Time
Happy	94	5	0	1	94	0.012 Sec.
Sad	1	98	0	1	98	0.013 Sec.
Angry	6	8	86	0	86	0,009 Sec.
Relax	0	18	1	81	81	0,009 Sec.
				Overall Average	**89,75**	**0,01 Sec**

Overall Average Prediction Time Across All Folders			**Overall Emotion Counts Summary Across All Folders**	
Happy	0,0096 Seconds		**Happy**	101
Sad	0,0101 Seconds		**Sad**	129
Angry	0,0097 Seconds		**Angry**	87
Relax	0,0105 Seconds		**Relax**	83

Following this, we examined the test results of our model using a different dataset. For testing, we selected the Panda et al. TAFFC Dataset - 2018 (http://mir.dei.uc.pt) as the evaluation dataset. This dataset contains 900 music files, each 30 seconds long, recorded at 22 kHz and 16-bit quality. The dataset was classified according to the Russel quadrant Q1, Q2, Q3, Q4 categories. Russell's Circumplex Model of Affect classifies human emotions based on two primary dimensions; arousal and valence. In this model, emotions are placed on a circular grid according to these two dimensions.

Valence dimension represents how pleasant or unpleasant an emotion is. Positive valence emotions include happiness and excitement, while negative valence emotions include sadness and anger.

Arousal dimension represents how activated or deactivated an emotion is. High arousal emotions include excitement and anger, while low arousal emotions include calmness and relaxation (1).

According to this model: High arousal and positive valence shows excitement, joy,happy.High arousal and negative valence shows anger, anxiety.Low arousal and

positive valence shows contentment, peacefulness,relax.Low arousal and negative valence shows sadness, boredom. We used same code inorder to test this new dataset.

The test results showed that each prediction took approximately 0.01 seconds and accurately predicted the emotional states of the songs with an accuracy rate of 44.50% (Table 7).

Table 7: Performance Analyze Of Optimized RFC Model Over ForeignDataset

Real Emotion	**Happy Prediction**	**Sad Prediction**	**Angry Prediction**	**Relax Prediction**	**Acc. %**	**Avarage Prediction Time**
Happy	176	32	14	3	78	0.011 Sec.
Sad	51	117	21	36	52	0.012 Sec.
Angry	177	3	43	2	19	0,010 Sec.
Relax	34	105	21	65	29	0,011 Sec.
				Overall Average	**44,50**	**0,011 Sec**

Overall Average Prediction Time Across All Folders			**Overall Emotion Counts Summary Across All Folders**	
Happy	0,0109 Seconds		**Happy**	438
Sad	0,0115 Seconds		**Sad**	257
Angry	0,0108 Seconds		**Angry**	99
Relax	0,0109 Seconds		**Relax**	106

CHAPTER V
CONCLUSION

The primary objective of this research was to develop a robust and efficient model for music emotion recognition, with a particular focus on Turkish music using deep neural networks. Through various modeling phases, including CNN, LSTM, and Random Forest Classifier architectures, the study achieved an accuracy of 81% on the Turkish dataset, highlighting the effectiveness of the proposed methodologies.

The results obtained from the Turkish dataset demonstrated a high accuracy rate comparable to models in the current literature. However, during the testing process, we observed a decrease in prediction accuracy when the model was applied to songs from other countries. We believe this discrepancy can be attributed to two main factors. First, the Turkish dataset used in our research was carefully reviewed and categorized into four distinct emotional states, while the test dataset was classified according to Russell's emotion model, which is based on the dimensions of arousal and valence. Second, since songs produced within popular culture often reflect the cultural context of their country of origin, we believe the differences in cultural expression between countries contributed to the variance in model performance.

These findings have significant implications for the field of music emotion recognition. By focusing on Turkish music, this research fills a gap in the literature, where studies on non-Western music datasets are limited. The developed model not only improves accuracy but also provides a framework that can be adapted to other cultural music datasets, potentially aiding in the development of culturally sensitive emotion recognition systems.

The signal modification methods employed in this study enabled the development of a highly accurate model with significantly lower costs. We believe that further research on foreign music, along with fine-tuning hyperparameters, could enhance the model's overall performance on non-Turkish music datasets. However, since the primary goal of this study was to develop a model with high predictive

accuracy for Turkish songs and capable of operating on portable devices, we believe our work effectively highlights the efficiency and effectiveness of our music emotion recognition approach, particularly in the context of Turkish music.

Finally, it is important to acknowledge the limitations of this study. The primary limitation was the small size of the dataset, which may have impacted the generalizability of the model. Additionally, the model's decreased performance on non-Turkish datasets suggests that cultural differences in music and emotion perception need further exploration and should be addressed in future studies.

As technology continues to evolve, the potential for improving the ability to accurately predict people’s emotional experiences while listening to music increases. Our study represents a step toward that understanding and we hope it will inspire further research into the complex relationships between music, culture, and emotion.

CHAPTER VI
FUTURE STUDIES

The method we follow in our current study is based on the classification of the physical properties of the songs in our dataset. The results we obtained and the model we developed were produced with this data. However, when we tested our entire dataset using our developed model, we found that the accuracy rates we achieved decreased when we tested it with datasets consisting of songs listened to in different countries. We think that this decrease in correct prediction rates is due to the differences in the cultural tendencies of the music listened to in different societies. In order to eliminate the decrease in the correct prediction rate and to increase the correct prediction rates of the model on songs listened to in different regions, we think that including metadata in the learning process, using song lyrics during modeling, and making changes to the settings of the hyperparameters used in the filtering and noise addition processes are the right strategies to follow. We believe that these approaches can increase the overall performance of the model and eliminate the effect of the observed cultural variability and achieve higher correct prediction rates. We hope that the findings of this study will provide valuable insights and support for future research in this field. We wish all academics working in this field success in their endavors.

REFERENCES

[1] MCFEE Brian, RAFFEL Colin, LIANG Dawen, ELLIS Daniel P.W., MCVIVAR Matt BATTENBERG Eric and NIETO Oriol (2015), "librosa: Audio and Music Signal Analysis in Python", *In, Proceedings Of The 14th Python in Science Conference*, Eds. HUFF Kathryn and BERGSTRA James, pp.18-24, SciPy, Texas, DOI:10.25080/Majora-7b98e3ed-003.

[2] RUSSEL James A. (1980), "A circumplex model of affect", *Journal of Personality and Social Psychology* Volume 39, Issue 6, pp.1161–78, DOI: 10.1037/h0077714.

[3] JUTHI Jannatul Humayra, GOMES Anthony, BHUIYAN Touhid and MAHMUD Imran (2020) "Music Emotion Recognition with the Extraction of Audio Features Using Machine Learning Approaches", *In, Proceedings of ICETIT 2019*, Eds. SINGH Pradeep Kumar, PANIGRAHI Bijaya Ketan, SURYADEVARA Nagender Kumar, SHARMA Sudhir Kumar and SINGH Amit Prakash, pp.318-329, Springer International Publishing, Cham, DOI: 10.1007/978-3-030-30577-2_27.

[4] CUNNINGHAM Stuart, WEINEL Jonathan and PICKING Richard (2018), "High-level analysis of audio features for identifying emotional valence in human singing", *In, Proceedings of The Audio Mostly 2018 on Sound in Immersion And Emotion*, Article No 37, pp. 1–4, Association for Computing Machinery, New York, USA, DOI: 10.1145/3243274.3243313.

[5] YADAV Ashima and VISHWAKARMA Dinesh (2020), "A multilingual framework of CNN and Bi-LSTM for emotion classification", *In, 11th International Conference on Computing, Communication and Networking Technologies (IC-CCNT)*, pp.1–6, DOI: 10.1109/icccnt49239.2020.9225614.

[6] BAGUS Atmaja Tris and MASATO Akagi (2020), "On The Differences Between Song and Speech Emotion Recognition: Effect of Feature Sets, Feature Types, and Classifiers", *2020 IEEE Region 10 Conference (TENCON)*, pp. 968-972, Osaka, Japan, DOI: 10.1109/TENCON50793.2020.9293852.

[7] SANTOS Arthur, KAREN Rosero Jácome and MASIERO Bruno (2021) “Song Emotion Recognition: A Study of the State of the Art”, *In, Anais do XVIII Simpósio Brasileiro de Computação Musical,* pp. 209-212, SBC Publishing, Porto Alegre, RS, Brasil, DOI: 10.5753/sbcm.2021.19449.

[8] HAN Donghong, KONG Yanru, HAN Jiayi, WANG Guoren (2022) “A survey of music emotion recognition”*, Frontiers of Computer Science,* Volume 16, Issue 6, p. 166335, DOI: 10.1007/s11704-021-0569-4.

[9] ER Mehmet Bilal and ESİN Emin Murat, (2021) “Music Emotion Recognition with Machine Learning Based on Audio Features”, *Computer Science*, Volume 6, Issue 3, pp. 133-144, DOI: 10.53070/bbd.945894

[10] BAI Junjie, LUO Kan, PENG Jun, SHI Jinliang, WU Ying, FENG Lixiao, LI Jianqing and WANG Yingxu (2017) “Music Emotions Recognition by Machine Learning With Cognitive Classification Methodologies”, *International Journal of Cognitive Informatics and Natural Intelligence* (IJCINI), Volume 11, Issue 4, pp. 80–92, DOI: 10.4018/ijcini.2017100105.

APPENDICES

APPENDIX A: DETAILED DATASET DESCRIPTION

Dataset Overview: In this study, Turkish Music Emotion Dataset [1], a dataset consisting entirely of Turkish songs, was used. The selected dataset contains songs with rich emotional content from different music genres, which can increase the generalization ability of the developed model. The selected dataset was meticulously selected and labeled to cover the wide emotional spectrum in Turkish music. The songs were divided into four emotional categories: happiness, sadness, anger, and relaxation.

There are a total of 400 songs, 100 of which are 30 seconds in each category. The original files in the dataset were encoded in MPEG Audio version 1 Layer 3 format with a variable bit rate of 192 Kb/s and a sampling rate of 44.1 kHz.

Preprocessing Steps: In its current form, compatibility problems were experienced during the analysis process while processing the audio files encoded in MP3 format. Therefore, a conversion process from MP3 to WAV format was applied. This change eliminated the compatibility problems. The detailed steps taken for preprocessing during the change in encoding format, including sampling rates, bit rates, and applied filters, are described in our study.

APPENDIX B: SIGNAL PROCESSING TECHIQUES

The Butterworth filter is a type of signal processing filter designed to have as flat a frequency response as possible in the passband. This filter eliminates the fluctuations in the passband. This is ideal for preserving the original signal while filtering out unwanted frequencies. The cutoff frequency parameter determines where the filter starts to attenuate the signal. Frequencies below the cutoff pass through the filter, while frequencies above it are gradually attenuated. The filter order parameter controls the steepness of the filter's rolloff. A higher order value means a steeper rolloff that separates the passband from the stopband more sharply. The Butterworth filter is ideal for our application because it maintains a flat response in the passband and suppresses high-frequency noise while preserving the emotional features of the music's lower frequencies. When used in this way, it is expected to improve the performance of our machine learning model by reducing irrelevant or noisy data in the feature extraction process. It is important to preserve the tonal and rhythmic elements that contribute to the emotional content of the music. The aim is to increase the model's accurate prediction performance by reducing noise and irrelevant details that may confuse the classifier. With this addition, it is possible to improve the performance of the model by filtering out unnecessary high-frequency components while preserving the emotional cues that are vital for musical emotion recognition.

Fourier Transform is a mathematical tool that transforms a signal from the time domain to the frequency domain. Fast Fourier Transform (FFT) is an algorithm that efficiently calculates the Discrete Fourier Transform (DFT). (FFT) helps us analyze the frequency components of a signal and is effectively and widely used in signal processing, especially for noise reduction, filtering and determining the dominant frequencies in a signal. Fourier Transform reveals the frequency content of a signal. A signal can be a combination of various sine waves at different frequencies, amplitudes and phases. Fourier Transform shows how much of each frequency is present in the signal by converting the signal into frequency components. This transformation is especially used when you want to filter out noise, identify patterns or understand the harmonic structure of a signal. In our study, we used Fast Fourier Transform (FFT) to convert the noisy signals in our audio files into clear sine waves. The Short Time Fourier Transform (STFT) is a variation of the Fourier Transform that analyzes a signal in small overlapping sections. Using this filter, we can understand how the frequency content of the signal changes over time, and this feature is useful for non-

stationary signals such as music or speech. Musical signals are inherently non-stationary, meaning their frequency content changes over time. The STFT provides a time-frequency representation that is essential for capturing the dynamic nature of music, such as changing pitch, rhythm, and timbre that contribute to emotional content. The magnitude of the STFT is often used as a basis for further feature extraction, such as Mel spectrograms, color features, or calculating spectral contrast. These features are critical for analyzing the emotional properties of music. Without using the Short Time Fourier Transform (STFT), we can directly plot the waveform of the audio signal before and after adding noise to show the effect of Gaussian noise in an audio signal. The graph below shows how the Gaussian noise added to the signal changes the signal in the time domain (Figure 19).

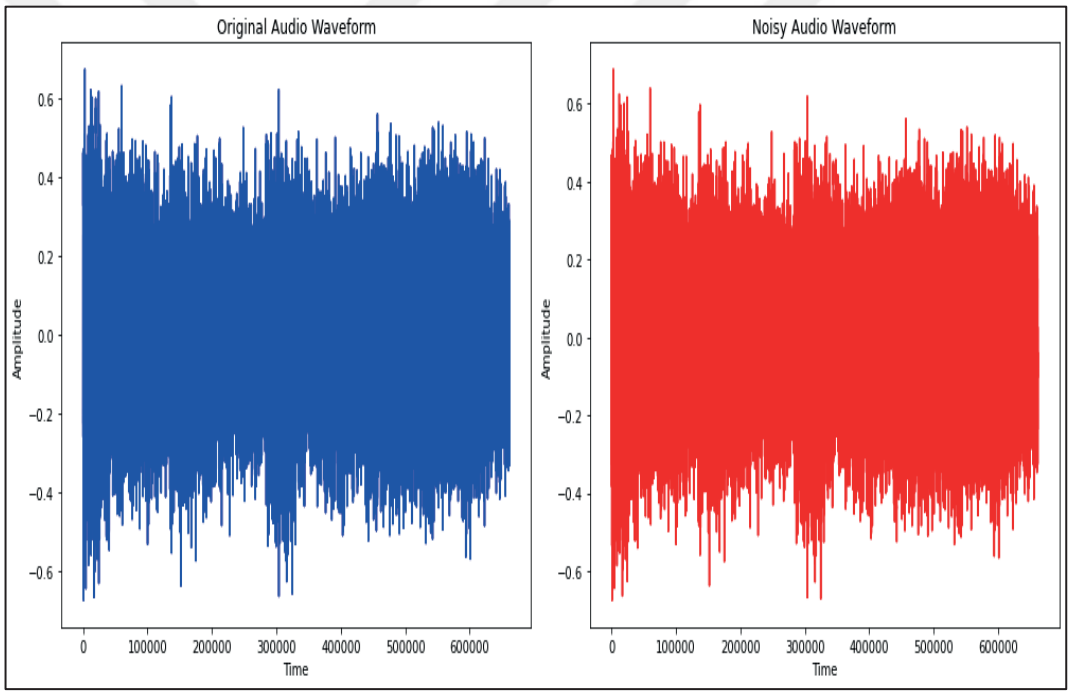

Figure 19: Plot Noise Alters The Signal In The Time Domain

Below are two graphs prepared to visually compare the Short Time Fourier Transform (STFT) of the audio signal of one of the music files used for the test before and after the addition of Gaussian noise. On the left side, you can examine the original STFT of the same song file, and on the right side, the STFT after the addition of noise. (Figure 20)

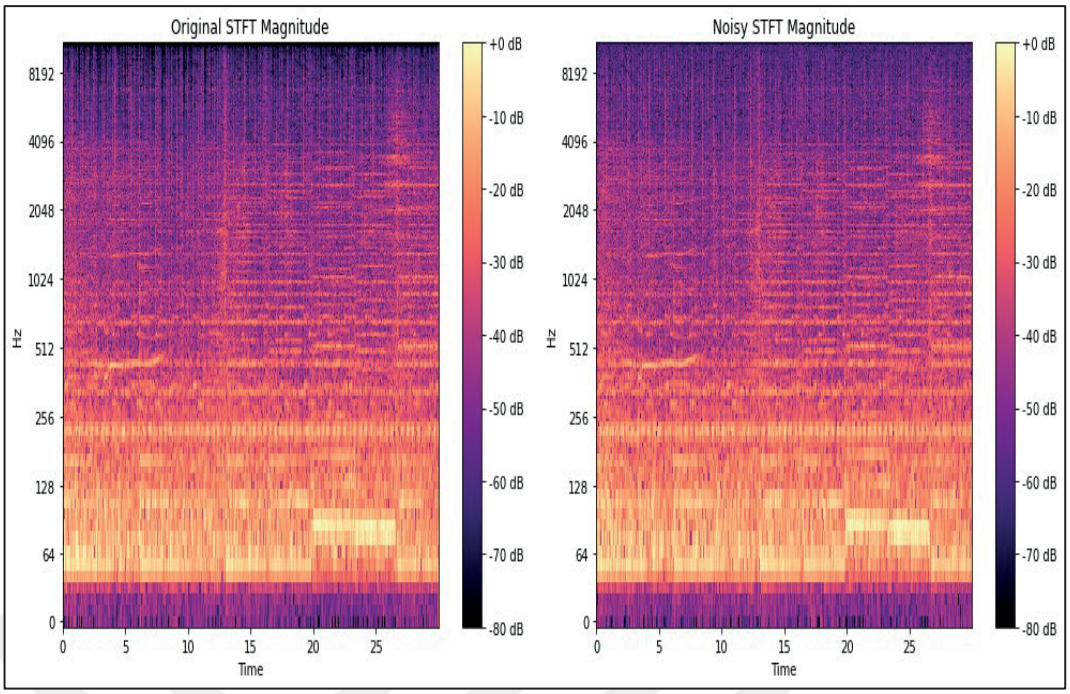

Figure 20: Plot STFT Application Effects Over Noisy Audio Files