

Music Genre Classification using CNN

A Project Report

Submitted by

**KARAN MEHTA,
DHRUV PATEL,
NIHAL SHETTY.**

Under the Guidance of
PROF. PRIYANKA VERMA

in partial fulfilment for the award of the degree
of
BACHELOR OF TECHNOLOGY CSBS
COMPUTER ENGINEERING

At



**MUKESH PATEL SCHOOL OF TECHNOLOGY
MANAGEMENT AND ENGINEERING**

OCTOBER, 2021.

Identification of Music Genre using Convolutional Neural Network

Agrawal Shreyash^{#1}, S. P. Dhanure^{#2}, P. P. Rathod^{#3}, Challawar Ashay^{#4}, Gadekar Pritesh^{#5}

^{1,2,3,4,5}Dept. of E&TC Engg., Smt. Kashibai Navale College of Engineering, Pune, Savitribai Phule Pune University, Pune

¹shreyash.agrawal.77198@gmail.com

²sudhir.dhanure_skncoe@sinhgad.edu

³pravinkumar.rathod_skncoe@sinhgad.edu

⁴ashaychallawar@gmail.com

⁵priteshgadekar26@gmail.com

Abstract—

This paper proposes the implementation of a Convolutional Neural Network (CNN) to classify music into its respective genre. The digital entertainment industry is booming right now and people are mostly listening to music online these days. Music plays a key role in our lives. The quantity of music being released on internet platforms is huge. But to manually classify music files is a hectic task for human beings. There is also a good chance of error in case of classification done by humans. The increase amount of work in this field recently has brought a great demand for automatic music genre classification. This paper presents a deep learning approach using Convolutional Neural Network (CNN). CNN model is trained using GTZAN dataset with ten musical genres and spectrogram of each audio file. The model essentially uses a neural specification to classify the music file into 10 different genres. CNN has good ability to work with various musical patterns. CNN has been mostly used in recent approaches and it is more efficient than standard machine learning approaches.

Keywords— Music genre classification, convolutional neural network, spectrogram, GTZAN, deep learning.

I. INTRODUCTION

Music performs an essential role in our lives. Music unites like-minded people and is a link between communities. Communities are usually identified by the types of songs that they have composed or may have heard. Different communities and groups listen different sorts of music. One main feature that separates one quite music from another is that the genre of the music.

As the number of songs keeps on increasing, people find it relatively hard to manage the songs of their taste. Since taking note of music online has become very convenient for people, because of the increase of online music streaming services like Spotify, iTunes, etc. users expect the music recommendation by the service. To make this possible, we need to check people's listening options and determine the genre they are listening to. This is the best way. Due to the rapid growth of the digital show business, automatic classification of music genres has acquired significant prominence in recent years. One way to effectively classify the song is based on genre. Classification of genre are often very valuable to elucidate some interesting problems like creating song references, tracking down related songs and music labelling.

II. LITERATURE SURVEY

Snigdha Chillara, Kavitha A. S., Shweta A. Neginhal, Shreya Haldia, Vidyullatha K. S. in [1] did a survey in music genre classification using machine learning algorithms, where comparison of different music genre classification techniques with machine and deep learning algorithms is given. Spectrogram-based models and Feature based models are used to find out the best model for

classification. CNN model was determined to be the best model over others like RNN, ANN, and LR. Subset of Free Music Archive [FMA] database is used. Librosa, a Python library is used for feature extraction purpose. Hareesh Bahuleyan in [2] proposed the work which gives an approach to classify music automatically, Comparison of the performance of two classes of models is done. CNN model is trained end-to-end using spectrogram and the second approach utilizes various ML algorithms like Logistic Regression, Random Forest, etc. VGG-16 CNN model gave highest accuracy of 89%. In this research 'Audio set' dataset is used which consists of mp4 files. The mp4 files are converted into the desired wav format. Tom LH Li, Antoni B. Chan, Andy HW. Chun in [3] made an effort to understand the main features which actually contribute to build the optimal model for Music Genre Classification. A methodology to automatically extract musical patterns features from audio music is been done. In this CNN has the strong capacity to capture informative features from the varying musical patterns. Due to this the approach is made to CNN, where the musical data have similar characteristics to image data. The dataset considered here is GTZAN which consists of 10 genres of 100 audio clips each. The musical patterns are evaluated using WEKA tool. Bryan Lansdown, Dr. Shan He in [4] gave comparison of machine learning algorithms in their suitedness to the task of music genre classification. Classification was based upon 54 features extracted from each song of the dataset, most of which pertained to spectral information. The average genre scores across the standard dataset were as follows: Electronic:60%, Folk: 73%, Hip-Hop: 74%, Pop: 41%, Rock: 73%. R. Thiruvengatanadhan in [5] proposed a technique of Music Genre Classification using GMM (Gaussian mixture model), a new technique that uses support vector machines to classify songs is described. Parametric or non-parametric methods are used to model the distribution of feature vectors. The music data is collected from music channels using a TV tuner card. A total dataset of 100 different songs is recorded, which is sampled at 22 kHz and encoded by 16-bit. In this work fixed length frames with duration of 20 ms and 50 percentages overlap (10 ms) are used. GMM method has good performance in musical genre classification scheme and is very effective and the accuracy rate is 94%.

III. PROPOSED SYSTEM

The proposed system helps classify music under the genres of the GTZAN dataset. Convolutional Neural Network (CNN) is a deep learning model which delivers great performance with images and basically saves sizeable number of efforts and time.

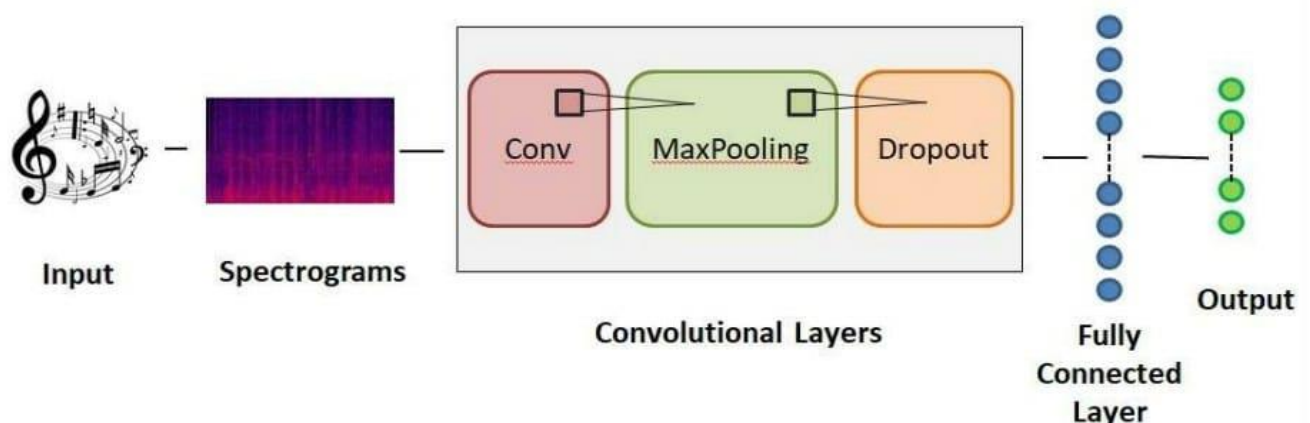


Fig. 1 Working of convolutional neural network

Input will be presented in the form of audio file in .wav format. Dataset used here to train and test the model is GTZAN which consist of 1000 different songs with 10 different genres (100 songs/genre). Each song will get converted to its respective spectrogram which is nothing but a 2-D representation of an audio signal, having time on x-axis and frequency on y-axis. The table provided below shows different genres in the GTZAN dataset.

TABLE I
GENRES IN GTZAN DATASET

Sr. No.	GENRES	No. of Songs
1	BLUES	100
2	CLASSICAL	100
3	POP	100
4	ROCK	100
5	COUNTRY	100
6	DISCO	100
7	HIP-HOP	100
8	JAZZ	100
9	REGGAE	100
10	METAL	100

The next step is to gather some parameters or features from the dataset. This step includes computing Mel spectrogram of every audio file in the dataset and also computing its MFCC(Mel Frequency Cepstral Coefficients). Mel spectrogram is generated using Mel scale. MFCC takes audio samples as input and after processing, it calculates coefficients unique to that particular sample. Also, several numeric features of the audio file are computed using Librosa library. The numeric features included are first 13 mfccs, zero-crossing rate, spectral centroid and spectral rolloff. This Mel spectrogram data and numeric features will be used to train and test the CNN model. Out of 1000, 800 songs are trained and 200 are tested.

Convolution Neural Network (CNN) is a deep learning module used to classify a short segment of the spectrogram, which is passed to hidden layers which consists of convolution, max pooling and dropout. The data is split into training and testing set of 800 and 200 songs respectively. CNN then predicts the computed genre of the music. CNN gave the training accuracy of 80% and testing accuracy of 62%. As number of epochs are increased, model learns more but too much epochs would lead to overfitting, so increasing the number of epochs would likely not improve the model. Convolutional neural network has its origin from the study of biological neural system. Humans can find and recognize patterns without having to re-learn the concept. Convolutional Neural Net can also do this because it recognizes an object as an object even when it appears in some different way.

IV. CNN ARCHITECTURE

At the heart of it all, convolutional neural networks can be seen as a neural network that uses the same copy of the same neuron. This allows the network to work with a lot of different hidden layers and perform on highly computational models. In simple terms, CNN can take an image as an input and it consists of multiple hidden layers of artificial neurons. Pre-Processing required in convolution networks is much lower as compared to other classification algorithms. As in primitive methods filters used are hand-engineered having enough training, convolution networks have the ability to automatically learn these filters. The result of this is specific features that can be detected anywhere

on input images. The architecture of this networks is similar to that of connectivity pattern of neurons in the Human Brain. The following figure shows how hidden layers work in CNN.

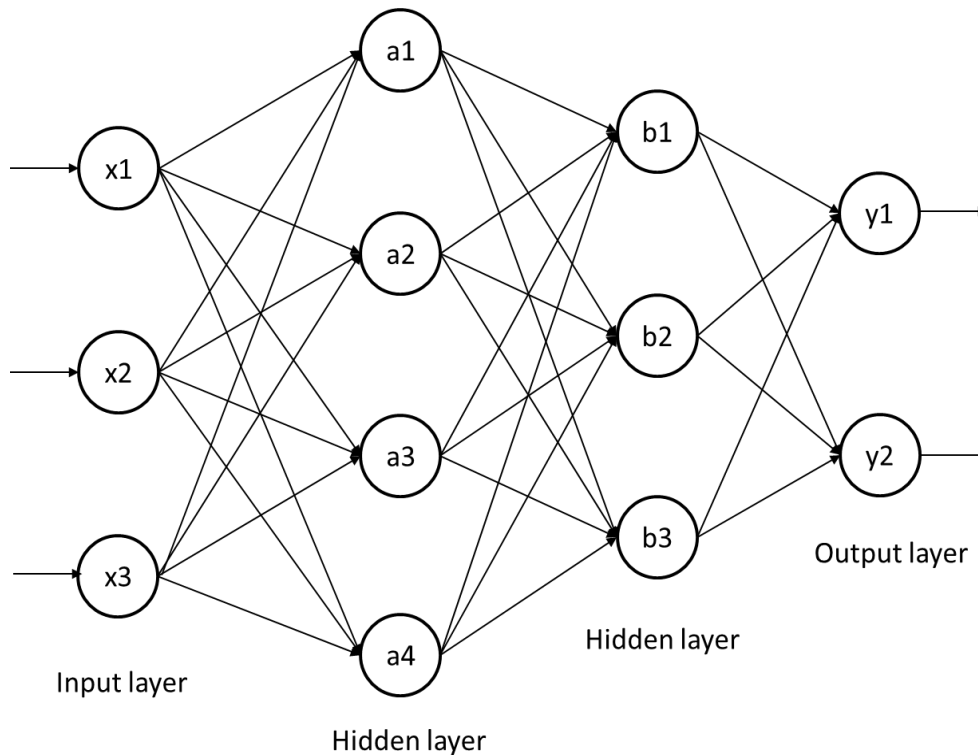


Fig 2 Layers of CNN

Convolutional layers are not only applied to input data but they can also be applied to the output of other layers. More data needs more layers to be able to learn more complicated patterns. In multiple layers the filters that work on raw data will extract low-level features and the extraction of features will be at high-level as depth of the network is increased. There are some basic and important layers that are used in CNN.

A. Convolution layer

This is the first layer of the network which is used to extract different features from the input spectrogramic image. In this layer all the mathematical operations of convolution is performed using the filters. In convolution layer the implementation starts with a larger image and ends with a smaller array that holds records of which sections of image were most interesting. In this the information about the image like the corners and edges is extracted.

B. Maxpooling and Dropout layer

Convolution layer is followed by pooling layer in most of the cases as the fundamental aim of this layer is to decrease the size of convolved feature and subsequently reduces the computational cost. This is done by reducing the connections between layers and independent operation on each feature map. This layer reduces the size of the array while keeping the most interesting features. Dropout layer is used just in case of over-fitting. To overcome this, few neurons are randomly dropped from the network which simultaneously reduces the size of model.

C. Fully-Connected layer

By convolution and maxpooling a large image is reduced in a small array. That array can be used as input to another layer named as fully connected layer. This layer is the final layer in which the main purpose of classification is carried out thus it is just before the final output.

The basic idea of these layers is to summarize a large image until the final result is achieved.

V. FUTURE SCOPE

This work was done for quick implementation so it has a lot of scope for improvement. The dataset used in this study has 1000 songs of 10 different genres. Training and testing using larger dataset in future might improve CNN's ability to classify music into its respective genre. Few songs result in weaker analysis and over fitting of some algorithms which can be improved with large dataset. Also using more features of the audio file could help to enhance the CNN accuracy.

VI. CONCLUSION

The proposed system is a technique for classifying music genres. The exponential growth within digital industry causes an urgent need for effective genre classification for users as well as content provider. The system used is robust and cost effective because open-source tools like Python, Jupyter can be used. And it is also easy to implement as it won't need much higher specifications. In case of classification done by human being there is a good chance of error because of the different nature of every individual. If we have some system which will categorize music into its respective genres automatically, we would save a large number of human efforts and time. CNN model used gives better accuracy i.e., 80% training accuracy than conventional machine learning algorithms. Misclassification may occur in some cases due to complex audio tracks but overall CNN gives better accuracy. So, to justify, it would be better to have an automatic genre classifier for music.

REFERENCES

- [1] Snigdha Chillara, Kavitha A S, Shweta A Neginhal, Shreya Haldia, Vidyullatha K S, "Music Genre Classification using Machine Learning Algorithms: A Comparison" International Research Journal of Engineering and Technology, Volume 06, Issue 05, May 2019.
- [2] Hareesh Bahuleyan, "Music Genre Classification using Machine Learning Techniques" University of Waterloo, Canada, Apr 2018.
- [3] Tom LH. Li, Antoni B. Chan, Andy HW. Chun, "Automatic Musical Pattern Feature Extraction using Convolutional Neural Network" International Multiconference of Engineers and Computer Scientists, Volume 01, March 2010.
- [4] Bryn Lansdown, Dr. Shan He. "Machine Learning for Music Genre Classification" University of Birmingham, September 2019.

- [5] George Tzanetakis, Perry Cook, "Music Genre Classification of Audio Signals" IEEE Transactions on speech and audio processing, Volume 10, Issue 05, July 2002.
- [6] S. Lippens, J. P. Martens, T. De Mulder, G. Tzanetakis, "A Comparison of Human and Automatic Musical Genre Classification." IEEE International Conference on Acoustics, Speech and Signal Processing, 2004.
- [7] Gabriel Gessle and Simon Akesson" A comparative analysis of CNN and LSTM for music genre classification" Degree Project in technology, Stockholm, Sweden 2019.
- [8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Journal of Machine Learning Research, 2014.
- [9] R. Thiruvengatanadhan "Music Genre Classification using GMM. (Gaussian mixture model)" (IRJET) volume 05 Issue:10 October 2018.
- [10] Gandharva Deshpande, Sushain Bhat, "Audio Genre Classification using Neural Networks" IRJET, Volume 06, Issue 08, August 2019.
- [11] Nirmal Vekariya, Hemang Vyas. Nirav Dedhiya, "Music Information Retrieval and Genre Classification using Machine and Deep Learning Techniques" IRJET, Volume 07, Issue 07, July 2020.
- [12] Weibin Zhang, Wenkang Lei, Xiangmin Xu, Xiaofeng Xing, "Improved Music Genre Classification using Convolutional Neural Networks" Interspeech 2016, September 2016.
- [13] Corey Kereliuk, Bob L. Sturm, Jan Larsen "Deep Learning and Music Adversaries" IEEE, Issue:2015.
- [14] Rajeeva Shreedhara Bhat, Rohit B. R., Mamatha K. R. "Music genre classification" IJCMS journal Volume 7, Issue 1, 2020.
- [15] Sam Clarke, Danny park, Audrien Guerard" Music Genre Classification Using Machine Learning Techniques"2012.
- [16] Tao Feng, "Deep learning for music genre classification", 2014.
- [17] Muhammad Asim Ali, Zain Ahmed Siddqui, "Automatic Music Genres Classification using Machine Learning", International Journal of Advanced Computer Science and Applications, Vol 8, No 8, 2017.
- [18] Meng A. Ahrendt, P. & Larsen J. "Improving music genre classification by short-time feature integration", ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, V, 497–500, 2005.
- [19] Michael I. Mandel and Daniel P.W. Ellis, "Song-level Features and Support Vector Machines for Music Classification", Queen Mary, University of London, 2005.
- [20] S. Sigstia and S. Dixon, "Improved music feature learning with deep neural networks," in Acoustics, Speech and Signal Processing (ICASSP), IEEE International Conference on IEEE, pp. 6959–6963, 2014.

Abstract

Music Genre classification is very important in today's world due to rapid growth in music tracks, both online and offline. In order to have better access to these we need to index them accordingly. Automatic music genre classification is important to obtain music from a large collection. Most of the current music genre classification techniques use machine learning techniques. In this report, we present a music dataset which includes ten different genres. A Deep Learning approach is used in order to train and classify the system. Here Convolution Neural Network is used for training and classification. Feature Extraction is the most crucial task for audio analysis. Mel Spectrogram is used as a feature vector for sound sample. The proposed system classifies music into various genres by extracting the feature vector. Our results show that the accuracy level of our system is around % and it will greatly improve and facilitate automatic classification of music genres.

Introduction

Machine Learning (ML) and specifically Artificial Intelligence (AI) have witnessed a monumental growth the development and in making machines or programs almost as capable and intelligent as human beings. Researchers, students, scientists as well as other enthusiasts of the field, are working on all kinds of aspects of the domain to make these ideas a possibility. The field of Computer Vision (CV) is also one of these.

The main objective of this Computer Vision is to develop programs/machines to see the real-world just as we humans do, to help them perceive these objects in a manner similar to ours if not identical, and to even use the information gained and insinuated for a plethora of applications such as Image Recognition, Analysis & Classification, Video Recognition, Analysis & Classification, Media Restoration and Retrieval, Recommendation Systems, Natural Language Processing (NLP), etc. The advancements made possible due to Computer Vision with Deep Learning (DL) have been constructed and perfected over time, primarily with the help of particular algorithm — a Convolutional Neural Network (CNN).

The impressive capabilities and the wide range of applications of CNN intrigued us to delve deeper into this topic and further improve our knowledge and experience by working on a project which involved the usage of CNN. The topic of Music Genre Recognition, Classifications, Prediction and Recommendation caught was found to be quite interesting to implement. With the help of the GTZAN dataset, which is a vast collection of 1000 audio files

(in .wav format) belonging to 10 genres, each being 30 seconds long, we decided to implement a CNN Model over the Mel Spectrograms images generated for each of the audio files.

Problem Statement

This project aims to predict the genre of an audio sample from a set of 10 defined classes by implementing a Convolutional Neural Network Model over the Mel Spectrograms generated from the GTZAN Dataset.

Results

Model No.	Dataset Size	Train/Test Splitting	Training Accuracy (%)	Validation Accuracy (%)
1.	10000	90/10	99.87	84
2.	1000	80/20	100	69.5
3.	1000	90/10	76.33	31

From the table we can see that when our model 3 was trained using the original dataset containing 1000 audio files, the validation accuracy is very low, i.e., 31. When the data was augmented and 10000 files were created from the 1000 files by segmenting the audio files, the validation accuracy of the model drastically improved to 84%. The predictions then made using this model were quite accurate which can be seen in our Jupyter notebook for model 1. Another model using 1000 random files (100 from each genre) was created and named Model 2. This model didn't work well with prediction because it had an average validation accuracy of 69.5%

Conclusion

The models we created are for Music genre classification. The exponential growth within digital industry causes an urgent need for effective genre classification for users as well as content provider. The model we created is robust and cost effective because open-source tools like Python, Jupyter can be used. And it is also easy to implement as it won't need much higher specifications. In case of classification done by human beings there is a good chance of error because of the different nature of every individual. If we have some system which will categorize music into its respective genres automatically, we would save a large number of human efforts and time. CNN model used gives better accuracies., 99.87% training accuracy and 84% validation accuracy, than conventional machine learning algorithms. Misclassification may occur in some cases due to complex audio tracks but overall CNN gives better accuracy.

Abstract

Music Genre classification is very important in today's world due to rapid growth in music tracks, both online and offline. In order to have better access to these we need to index them accordingly. Automatic music genre classification is important to obtain music from a large collection. Most of the current music genre classification techniques uses machine learning techniques. In this paper, we present a music dataset which includes ten different genres. A Deep Learning approach is used in order to train and classify the system. Here convolution neural network is used for training and classification. Feature Extraction is the most crucial task for audio analysis. Mel Frequency Cepstral Coefficient (MFCC) is used as a feature vector for sound sample. The proposed system classifies music into various genres by extracting the feature vector. Our results show that the accuracy level of our system is around 76% and it will greatly improve and facilitate automatic classification of music genres.

Dataset Used

For this project, the dataset that we will be working with is the GTZAN Music Genre Classification dataset which consists of 1,000 audio tracks, each 30 seconds long. It contains 10 genres, each represented by 100 tracks. The 10 genres are Blues, Classical, Country, Disco, Hip-hop, Jazz, Metal, Pop, Reggae, Rock. The dataset has the following folders:

- Genres original — A collection of 10 genres with 100 audio files each, all having a length of 30 seconds (the famous GTZAN dataset, the MNIST of sounds)
- Images original — A visual representation for each audio file. One way to classify data is through neural networks because NN's usually take in some sort of image representation.
- 2 CSV files — Containing features of the audio files. One file has for each song (30 seconds long) a mean and variance computed over multiple features that can be extracted from an audio file. The other file has the same structure, but the songs are split before into 3 seconds audio files.

GTZAN Dataset with CSV- <https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification>
(<https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification>)

Original GTZAN Dataset- <http://marsyas.info/downloads/datasets.html>
(<http://marsyas.info/downloads/datasets.html>)

Installing necessary libraries

(Librosa is used for audio data)

In [1]:

```
!pip install librosa
```

In [2]:

```
!pip install python_speech_features
```

In [3]:

```
!pip install tensorflow
```

In [2]:

```
!pip install pydot
```

Collecting pydot

Downloading pydot-1.4.2-py2.py3-none-any.whl (21 kB)

Requirement already satisfied: pyparsing>=2.1.4 in c:\users\admin\anaconda3\lib\site-packages (from pydot) (2.4.7)

Installing collected packages: pydot

Successfully installed pydot-1.4.2

In [6]:

```
!pip install pydub
```

Collecting pydub

Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)

Installing collected packages: pydub

Successfully installed pydub-0.25.1

Importing Libraries

In [61]:

```
import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import matplotlib.pyplot as plt
import matplotlib.cm as cm
from matplotlib.colors import Normalize
import scipy
import seaborn as sns
%matplotlib inline

import librosa
import librosa.display
import IPython.display as ipd
from IPython.display import Audio

import warnings
warnings.filterwarnings('ignore')

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow import keras
from python_speech_features import mfcc
import random

from scipy import misc
import glob
from PIL import Image
from keras import layers
from keras.layers import (Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization,
                           Conv2D, AveragePooling2D, MaxPooling2D, GlobalMaxPooling2D, Dropout)
from keras.models import Model, load_model
from keras.preprocessing import image
from keras.utils import layer_utils
import pydot
from datetime import datetime
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
from keras.utils.vis_utils import plot_model
from tensorflow.keras.optimizers import Adam
from keras.initializers import glorot_uniform
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from pydub import AudioSegment
import shutil
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
import random
```

Understanding the Audio Files

In [20]:

```
#Understanding Audio
# Importing 1 file
y, sr = librosa.load(f'{location}/classical/classical.00024.wav')

print('y: ', y, '\n')
print('y shape:', np.shape(y), '\n')
print('Sample Rate (KHz):', sr, '\n')

# Verify Length of the audio
print('Check Len of Audio:', 661794/22050)
```

```
y: [-0.03775024 -0.07223511 -0.08242798 ...  0.02264404  0.00308228
    -0.0105896 ]
```

```
y shape: (661794,)
```

```
Sample Rate (KHz): 22050
```

```
Check Len of Audio: 30.013333333333332
```

In [21]:

```
df=df.drop(labels="filename",axis=1)
```

In [64]:

```
audio_recording="C:/Users/Admin/Desktop/ML Project/Datasets/genres/classical/classical.00024.wav"
data,sr=librosa.load(audio_recording)
print(type(data),type(sr))
```

```
<class 'numpy.ndarray'> <class 'int'>
```

In [65]:

```
librosa.load(audio_recording,sr)
```

Out[65]:

```
(array([-0.03775024, -0.07223511, -0.08242798, ...,  0.02264404,
        0.00308228, -0.0105896 ], dtype=float32),
 22050)
```

Playing Audio

In [24]:

```
import IPython
IPython.display.Audio(data,rate=sr)
```

Out[24]:

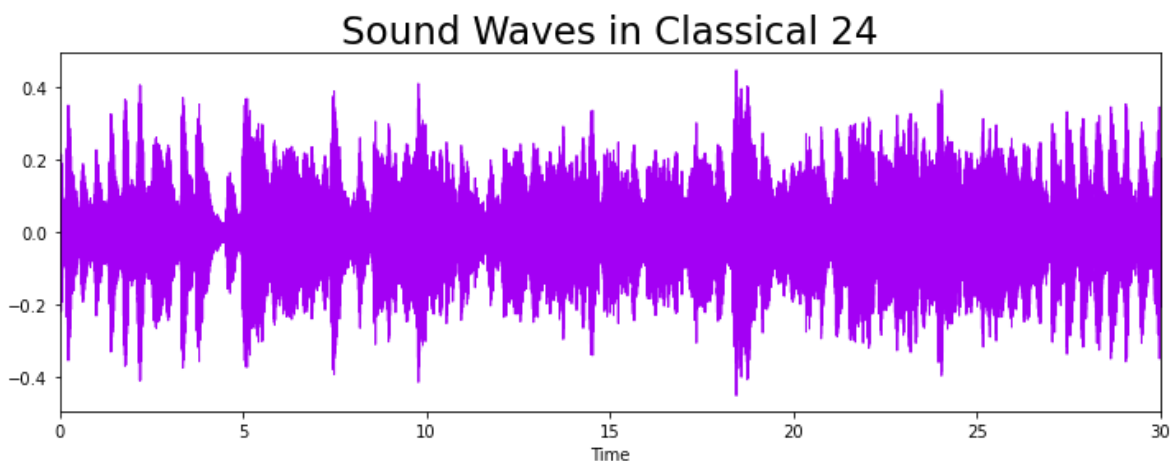


Waveforms

Waveforms are visual representations of sound as time on the x-axis and amplitude on the y-axis. They are great for allowing us to quickly scan the audio data and visually compare and contrast which genres might be more similar than others.

In [25]:

```
plt.figure(figsize=(12,4))
librosa.display.waveplot(data,color="#A300F4")
plt.title("Sound Waves in Classical 24", fontsize = 23);
plt.show()
```



Mel Spectrogram

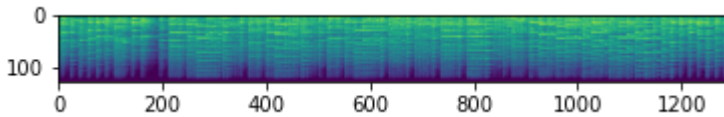
Mel spectrogram is a spectrogram that is converted to a Mel scale. The Mel scale mimics how the human ear works, with research showing humans don't perceive frequencies on a linear scale. Humans are better at detecting differences at lower frequencies than at higher frequencies.

In [66]:

```
mels = librosa.feature.melspectrogram(data)
fig = plt.figure()
canvas = FigureCanvas(fig)
p = plt.imshow(librosa.power_to_db(mels, ref=np.max))
p
```

Out[66]:

<matplotlib.image.AxesImage at 0x24ad8bf5700>



MFCC

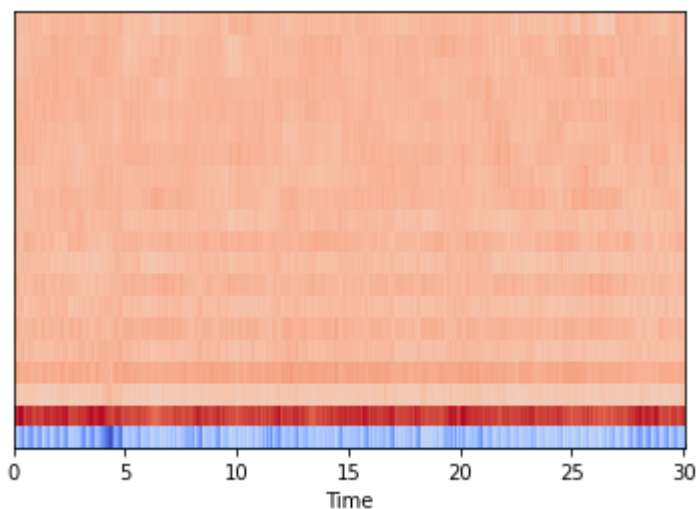
The Mel frequency cepstral coefficients (MFCCs) of a signal are a small set of features (usually about 10-20) that concisely describe the overall shape of a spectral envelope. It models the characteristics of the human voice. `librosa.feature.mfcc` computes MFCCs across an audio signal:

In [26]:

```
#using classical.00024.wav to display its MFCC
x, fs = librosa.load("C:/Users/Admin/Desktop/ML Project/Datasets/genres/classical/classical
librosa.display.waveplot(x, sr=sr)
mfccs = librosa.feature.mfcc(x, sr=fs)
(13, 97)
#Displaying the MFCCs:
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
```

Out[26]:

<matplotlib.collections.QuadMesh at 0x25a8a0d96a0>



Spectrogram

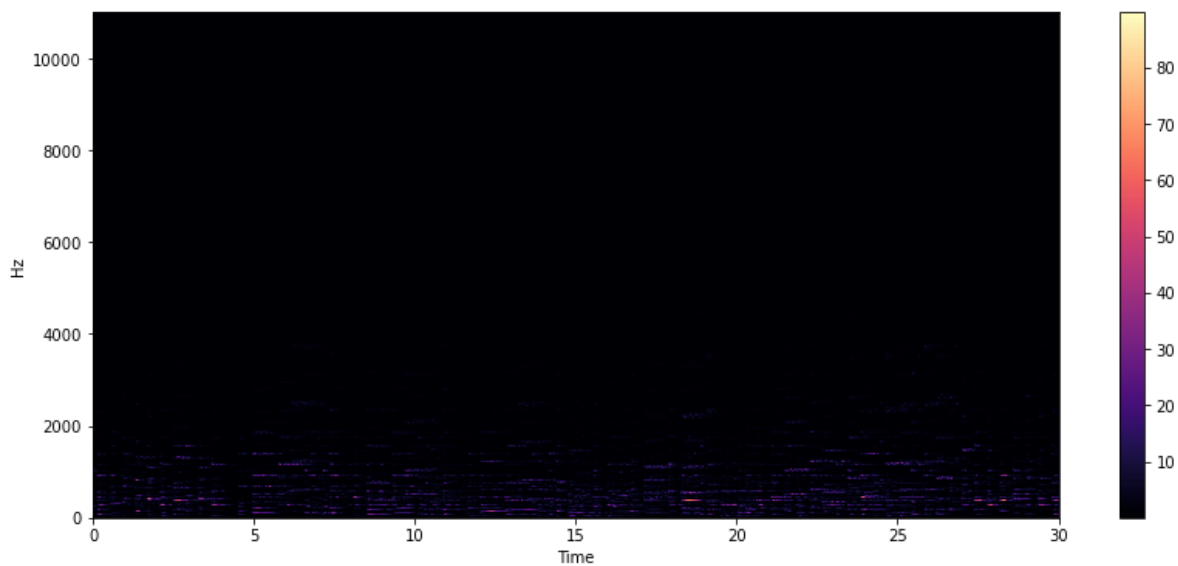
A spectrogram is a visual way of representing the signal loudness of a signal over time at various frequencies present in a particular waveform. Not only can one see whether there is more or less energy at, for example, 2 Hz vs 10 Hz, but one can also see how energy levels vary over time. Spectrograms are sometimes called sonographs, voiceprints, or voicegrams. When the data is represented in a 3D plot, they may be called waterfalls. In 2-dimensional arrays, the first axis is frequency while the second axis is time

In [27]:

```
stft=librosa.stft(data)
stft_db=librosa.amplitude_to_db(abs(stft))
plt.figure(figsize=(14,6))
librosa.display.specshow(stft,sr=sr,x_axis='time',y_axis='hz')
plt.colorbar()
```

Out[27]:

<matplotlib.colorbar.Colorbar at 0x25a8c49e790>

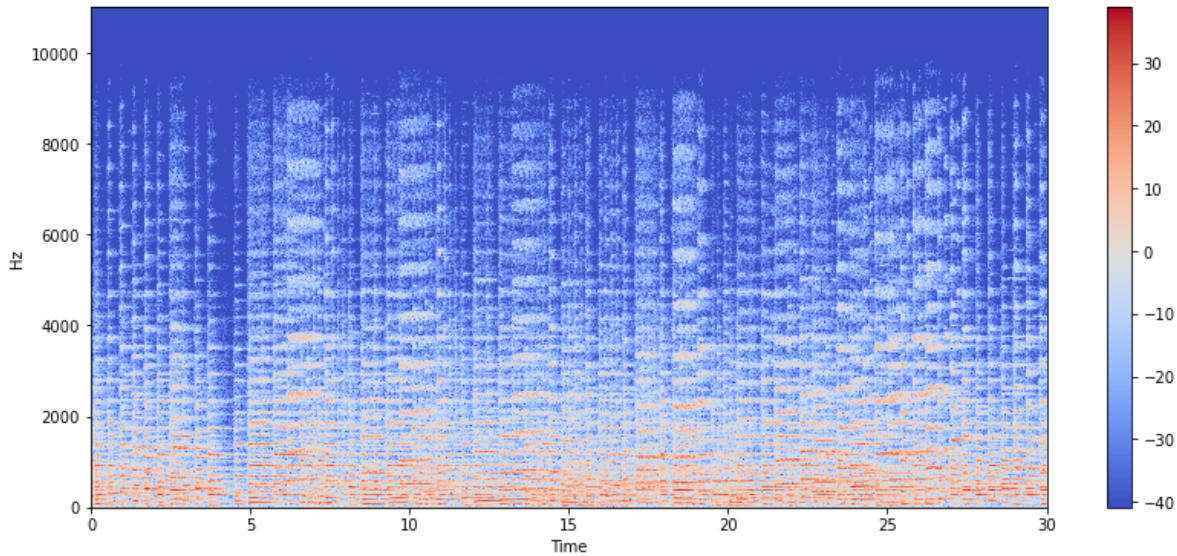


In [28]:

```
stft=librosa.stft(data)
stft_db=librosa.amplitude_to_db(abs(stft))
plt.figure(figsize=(14,6))
librosa.display.specshow(stft_db,sr=sr,x_axis='time',y_axis='hz')
plt.colorbar()
```

Out[28]:

<matplotlib.colorbar.Colorbar at 0x25a8eb79460>



Spectral Rolloff

Spectral Rolloff is the frequency below which a specified percentage of the total spectral energy, e.g. 85%, lies. `librosa.feature.spectral_rolloff` computes the rolloff frequency for each frame in a signal.

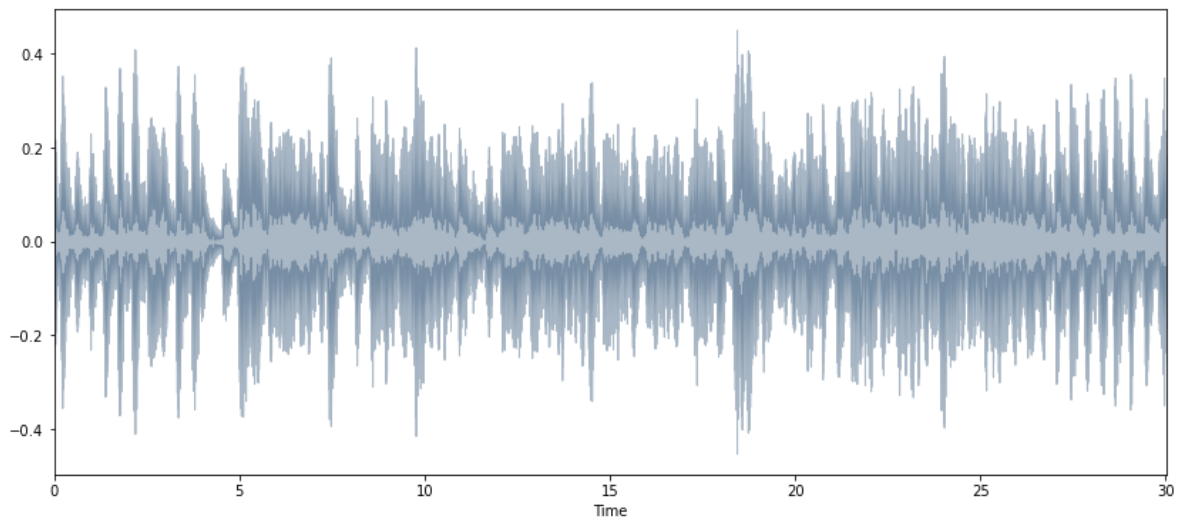
The roll-off frequency denotes the approximate low bass and high treble limits in a frequency response curve, with all frequencies between being those a speaker will play accurately.

In [29]:

```
spectral_rolloff=librosa.feature.spectral_rolloff(data+0.01,sr=sr)[0]  
plt.figure(figsize=(14,6))  
librosa.display.waveplot(data,sr=sr,alpha=0.4,color="#2B4F72")
```

Out[29]:

<matplotlib.collections.PolyCollection at 0x25a8ebdf520>

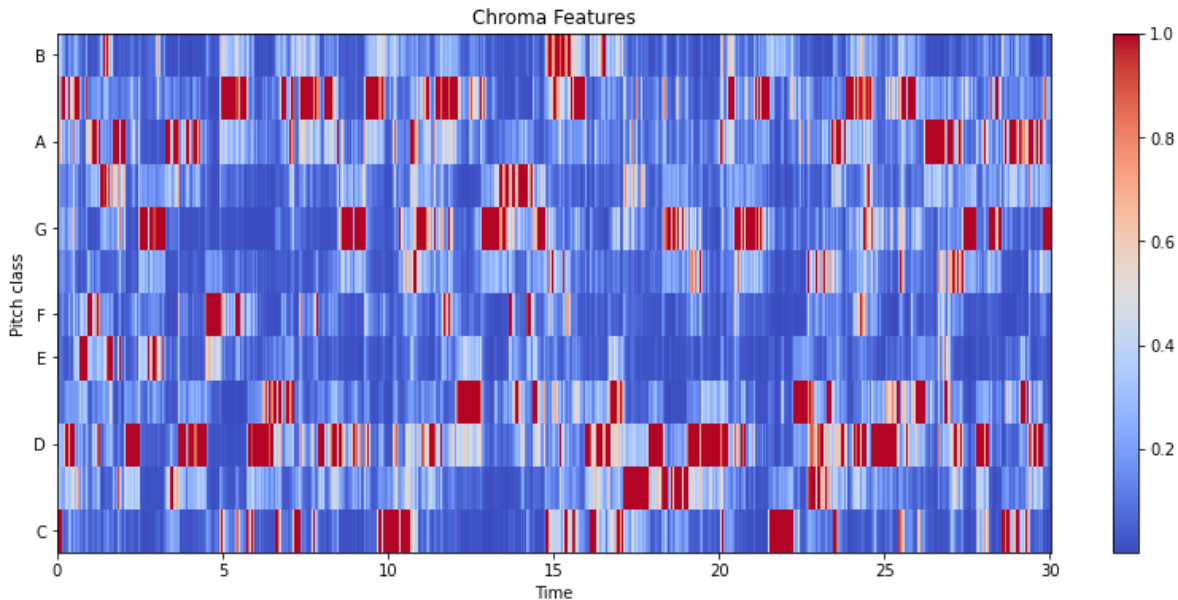


Chroma Feature

It's a useful tool for studying musical aspects whose pitches may be classified and whose tuning is close to the equal-tempered scale. Chromatic and melodic aspects of music are captured by chroma features, which are resistant to changes in timbre and instrumentation.

In [30]:

```
import librosa.display as lplt
chroma = librosa.feature.chroma_stft(data,sr=sr)
plt.figure(figsize=(14,6))
lplt.specshow(chroma,sr=sr,x_axis="time",y_axis="chroma",cmap="coolwarm")
plt.colorbar()
plt.title("Chroma Features")
plt.show()
```

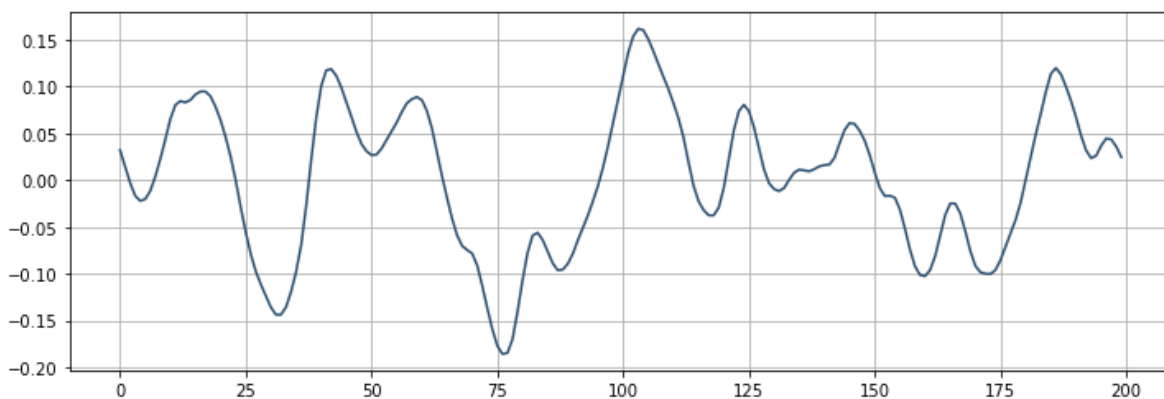


Zero Crossing

Zero crossing is said to occur if successive samples have different algebraic signs. The rate at which zero-crossings occur is a simple measure of the frequency content of a signal. Zero-crossing rate is a measure of the number of times in a given time interval/frame that the amplitude of the speech signals passes through a value of zero.

In [31]:

```
start=1000
end=1200
plt.figure(figsize=(12,4))
plt.plot(data[start:end],color="#2B4F72")
plt.grid()
```



In [32]:

```
zero_cross_rate=librosa.zero_crossings(data[start:end],pad=False)
print("the number of zero_crossings is :", sum(zero_cross_rate))
```

the number of zero_crossings is : 12

Therefore in our project, we have decided to generate Mel Spectrograms for all the audio files for the CNN Model

In [10]:

```
#Displaying the classes
location = 'C:/Users/Admin/Desktop/ML Project/Datasets/genres'
print(list(os.listdir(f'{location}')))
```

```
['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop',
'reggae', 'rock']
```

In [13]:

```
os.makedirs('C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms3sec')
os.makedirs('C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms3sec/train')
os.makedirs('C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms3sec/test')
```

In [41]:

```
genres = 'blues classical country disco hiphop jazz metal pop reggae rock'
genres = genres.split()
```

In [21]:

```
#Creating folders for the 3sec audio files and their spectrograms
for g in genres:
    path_audio = os.path.join('C:/Users/Admin/Desktop/ML Project/Datasets/audio3sec',f'{g}')
    os.makedirs(path_audio)
    path_train = os.path.join('C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms3sec/train',f'{g}')
    path_test = os.path.join('C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms3sec/test',f'{g}')
    os.makedirs(path_train)
    os.makedirs(path_test)
```

In [67]:

```

from pydub import AudioSegment
i = 0
for g in genres:
    j=0
    print(f"{g}")
    for filename in os.listdir(os.path.join('C:/Users/Admin/Desktop/ML Project/Datasets/genre

        song = os.path.join(f'C:/Users/Admin/Desktop/ML Project/Datasets/genres/{g}',f'{filen
        j = j+1
        for w in range(0,10):
            i = i+1
            #pydub calculates in milliseconds so we multiply the values by 1000
            t1 = 3*(w)*1000
            t2 = 3*(w+1)*1000
            newAudio = AudioSegment.from_wav(song)
            new = newAudio[t1:t2]
            new.export(f'C:/Users/Admin/Desktop/ML Project/Datasets/audio3sec/{g}/{g+str(j)+str(w

```

```

blues
classical
country
disco
hiphop
jazz
metal
pop
reggae
rock

```

In [68]:

```

for g in genres:
    j = 10
    print(g)
    for filename in os.listdir(os.path.join('C:/Users/Admin/Desktop/ML Project/Datasets/audio3sec', f'{g}')):
        song = os.path.join(f'C:/Users/Admin/Desktop/ML Project/Datasets/audio3sec/{g}', f'{filename}')
        y, sr = librosa.load(song, duration=3)
        mels = librosa.feature.melspectrogram(y=y, sr=sr)
        fig = plt.figure()
        canvas = FigureCanvas(fig)
        p = plt.imshow(librosa.power_to_db(mels, ref=np.max))
        melspecname = (f'C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms3sec/train/{g}/{filename}.png')
        plt.savefig(f'{melspecname}.png', format="png")
    j = j+1

```

blues
classical
country
disco
hiphop
jazz
metal
pop
reggae
rock

<Figure size 432x288 with 0 Axes>

Splitting the Dataset into a 90:10 split

In [42]:

```

directory = "C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms3sec/train/"
for g in genres:
    filenames = os.listdir(os.path.join(directory, f'{g}'))
    random.shuffle(filenames)
    test_files = filenames[0:100]

    for f in test_files:
        shutil.move(directory + f'{g}' + "/" + f, "C:/Users/Admin/Desktop/ML Project/Datasets/spe

```

Using Data Generators to further augment the dataset

In [44]:

```
train_dir = "C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms3sec/train/"
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(train_dir,target_size=(288,432),color_m

validation_dir = "C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms3sec/test/"
vali_datagen = ImageDataGenerator(rescale=1./255)
vali_generator = vali_datagen.flow_from_directory(validation_dir,target_size=(288,432),colo
```

Found 9000 images belonging to 10 classes.
Found 1000 images belonging to 10 classes.

In [45]:

```
#defining class labels for prediction
class_labels = ['blues',
                'classical',
                'country',
                'disco',
                'hiphop',
                'jazz',
                'metal',
                'pop',
                'reggae',
                'rock']
```

In [46]:

```
def GenreModel(input_shape = (288,432,4),classes=10):
    np.random.seed(10)
    X_input = Input(input_shape)

    X = Conv2D(8,kernel_size=(3,3),strides=(1,1),kernel_initializer = glorot_uniform(seed=10))
    X = BatchNormalization(axis=3)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((2,2))(X)

    X = Conv2D(16,kernel_size=(3,3),strides = (1,1),kernel_initializer=glorot_uniform(seed=10))
    X = BatchNormalization(axis=3)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((2,2))(X)

    X = Conv2D(32,kernel_size=(3,3),strides = (1,1),kernel_initializer = glorot_uniform(seed=10))
    X = BatchNormalization(axis=3)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((2,2))(X)

    X = Conv2D(64,kernel_size=(3,3),strides=(1,1),kernel_initializer=glorot_uniform(seed=10))
    X = BatchNormalization(axis=-1)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((2,2))(X)

    X = Flatten()(X)

    X = Dropout(rate=0.3)

    X = Dense(classes, activation='softmax', name='fc' + str(classes), kernel_initializer = glorot_uniform(seed=10))

    model = Model(inputs=X_input,outputs=X,name='GenreModel')

    return model
```

In [47]:

```
def plotValidate(history):
    print("Validation Accuracy",max(history.history["val_accuracy"]))
    pd.DataFrame(history.history).plot(figsize=(12,6))
    plt.show()
```


In [48]:

```
def plot_history(hist):
    plt.figure(figsize=(20,15))
    fig, axs = plt.subplots(2)
    fig.tight_layout(pad=2.0)

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

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

    plt.show()
```

In [49]:

```

import keras.backend as K
def get_f1(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val

model = GenreModel(input_shape=(288,432,4),classes=10)
opt = Adam(learning_rate=0.0005)
model.compile(optimizer = opt,loss='categorical_crossentropy',metrics=['accuracy',get_f1])

history = model.fit_generator(train_generator,epochs=70,validation_data=vali_generator)

```

```

Epoch 53/70
71/71 [=====] - 514s 7s/step - loss: 0.0065 - ac
curacy: 0.9989 - get_f1: 0.9989 - val_loss: 0.7283 - val_accuracy: 0.8340
- val_get_f1: 0.8389
Epoch 54/70
71/71 [=====] - 516s 7s/step - loss: 0.0040 - ac
curacy: 0.9993 - get_f1: 0.9993 - val_loss: 0.7262 - val_accuracy: 0.8450
- val_get_f1: 0.8412
Epoch 55/70
71/71 [=====] - 520s 7s/step - loss: 0.0124 - ac
curacy: 0.9980 - get_f1: 0.9980 - val_loss: 0.7676 - val_accuracy: 0.8320
- val_get_f1: 0.8330
Epoch 56/70
71/71 [=====] - 519s 7s/step - loss: 0.0084 - ac
curacy: 0.9984 - get_f1: 0.9985 - val_loss: 0.7332 - val_accuracy: 0.8330
- val_get_f1: 0.8354
Epoch 57/70
71/71 [=====] - 531s 7s/step - loss: 0.0054 - ac
curacy: 0.9988 - get_f1: 0.9988 - val_loss: 0.8048 - val_accuracy: 0.8280
- val_get_f1: 0.8275

```

Model Results

In [50]:

```
model.summary()
```

Model: "GenreModel"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 288, 432, 4)]	0
conv2d (Conv2D)	(None, 286, 430, 8)	296
batch_normalization (BatchNo	(None, 286, 430, 8)	32
activation (Activation)	(None, 286, 430, 8)	0
max_pooling2d (MaxPooling2D)	(None, 143, 215, 8)	0
conv2d_1 (Conv2D)	(None, 141, 213, 16)	1168
batch_normalization_1 (Batch	(None, 141, 213, 16)	64
activation_1 (Activation)	(None, 141, 213, 16)	0
max_pooling2d_1 (MaxPooling2	(None, 70, 106, 16)	0
conv2d_2 (Conv2D)	(None, 68, 104, 32)	4640
batch_normalization_2 (Batch	(None, 68, 104, 32)	128
activation_2 (Activation)	(None, 68, 104, 32)	0
max_pooling2d_2 (MaxPooling2	(None, 34, 52, 32)	0
conv2d_3 (Conv2D)	(None, 32, 50, 64)	18496
batch_normalization_3 (Batch	(None, 32, 50, 64)	256
activation_3 (Activation)	(None, 32, 50, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 16, 25, 64)	0
flatten (Flatten)	(None, 25600)	0
fc10 (Dense)	(None, 10)	256010
=====		
Total params: 281,090		
Trainable params: 280,850		
Non-trainable params: 240		

In [51]:

```
history.history
```

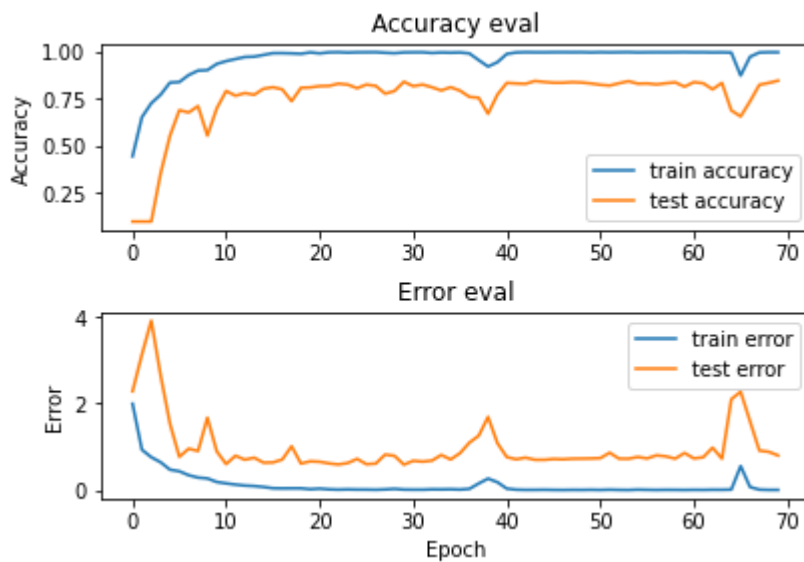
Out[51]:

```
{'loss': [1.998122215270996,  
 0.9346068501472473,  
 0.7625085115432739,  
 0.6451578736305237,  
 0.47415417432785034,  
 0.4365347921848297,  
 0.3457659184932709,  
 0.29043489694595337,  
 0.2711506485939026,  
 0.1893102377653122,  
 0.15677066147327423,  
 0.12895430624485016,  
 0.10626860707998276,  
 0.09508618712425232,  
 0.0738464817404747,  
 0.0444365069270134,  
 0.04107418656349182,  
 0.04236818850040436.]
```

In [52]:

```
plot_history(history)
```

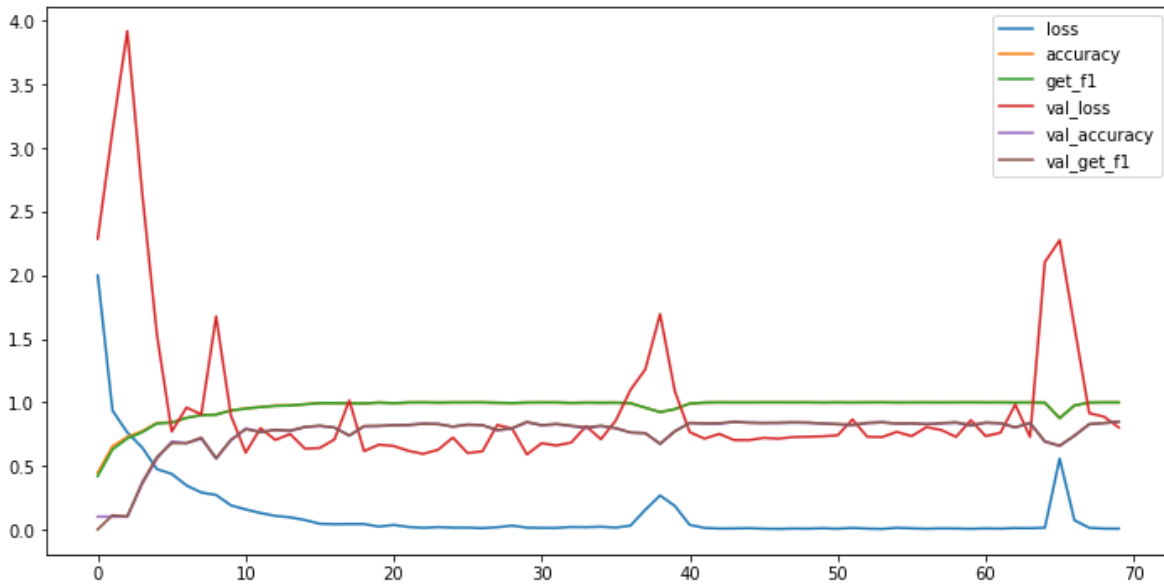
<Figure size 1440x1080 with 0 Axes>



In [53]:

```
plotValidate(history)
```

Validation Accuracy 0.8479999899864197



Predicting the Genre of a random song using the Model

In [54]:

```
def convert_mp3_to_wav(music_file):
    sound = AudioSegment.from_mp3(music_file)
    sound.export("music_file.wav", format="wav")
```

In [55]:

```
def extract_relevant(s,t1,t2):
    wav_file = os.path.join(f'C:/Users/Admin/Desktop/ML Project/Datasets/sample/',f'{s}')
    wav = AudioSegment.from_wav(wav_file)
    wav = wav[1000*t1:1000*t2]
    wav.export("extracted.wav", format='wav')
```

In [56]:

```
def create_melspectrogram(wav_file):
    y, sr = librosa.load(wav_file, duration=3)
    mels = librosa.feature.melspectrogram(y=y, sr=sr)

    fig = plt.figure()
    canvas = FigureCanvas(fig)
    p = plt.imshow(librosa.power_to_db(mels, ref=np.max))
    plt.savefig('melspectrogram.png')
```

In [57]:

```
def predict(image_data, model):

    image = img_to_array(image_data)

    image = np.reshape(image, (1, 288, 432, 4))

    prediction = model.predict(image/255)

    prediction = prediction.reshape((10,))

    class_label = np.argmax(prediction)

    return class_label, prediction
```

In [58]:

```
def show_output(songname):
    extract_relevant(songname, 40, 50)
    create_melspectrogram("extracted.wav")
    image_data = load_img('melspectrogram.png', color_mode='rgba', target_size=(288, 432))

    class_label, prediction = predict(image_data, model)

    print("## The Genre of Song is ", class_labels[class_label])

    prediction = prediction.reshape((10,))

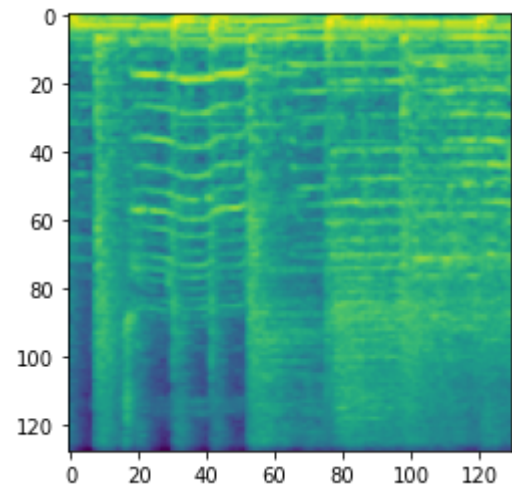
    color_data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    my_cmap = cm.get_cmap('jet')
    my_norm = Normalize(vmin=0, vmax=10)

    fig, ax = plt.subplots(figsize=(6, 4.5))
    ax.bar(x=class_labels, height=prediction,
          color=my_cmap(my_norm(color_data)))
    plt.xticks(rotation=45)
    ax.set_title("Probability Distribution Of The Given Song Over Different Genres")
    plt.show()
```

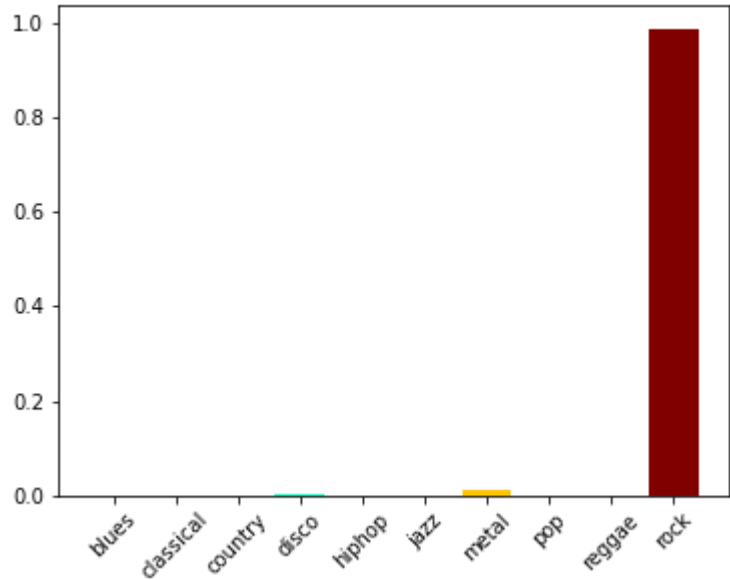
In [62]:

```
song = "NirvanaSmellsLikeTeenSpirit.wav"  
show_output(song)
```

The Genre of Song is rock



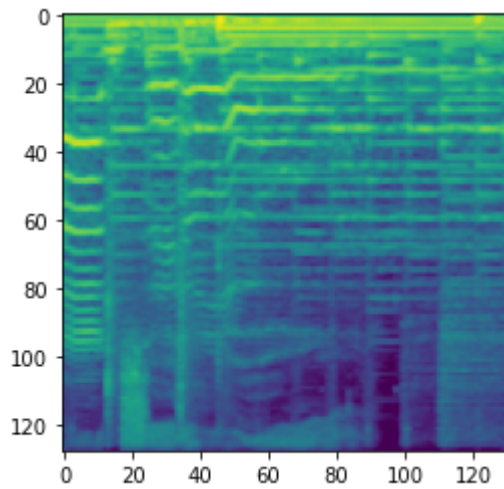
Probability Distribution Of The Given Song Over Different Genres



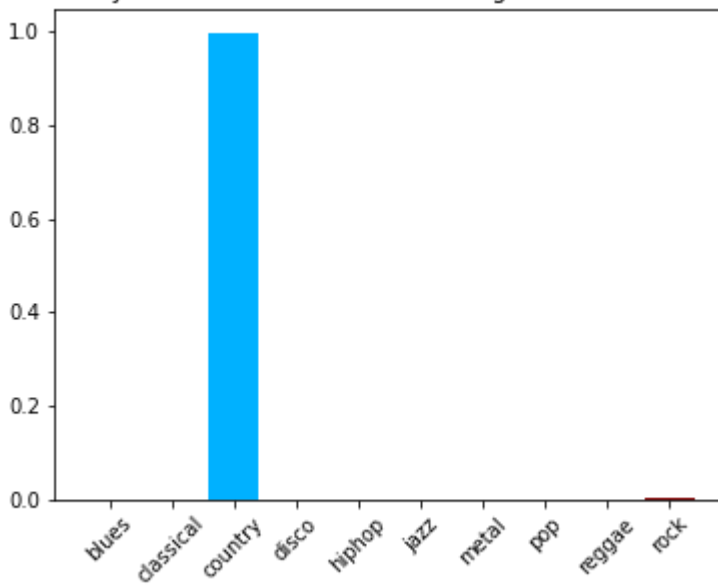
In [63]:

```
song = "TaylorSwiftLoveStory.wav"  
show_output(song)
```

The Genre of Song is country



Probability Distribution Of The Given Song Over Different Genres



In []:

Installing necessary libraries

(Librosa is used for audio data)

In []:

```
!pip install librosa
```

In []:

```
!pip install python_speech_features
```

In []:

```
!pip install tensorflow
```

In []:

```
!pip install pydot
```

In []:

```
!pip install pydub
```

Importing Libraries

In [1]:

```

import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import matplotlib.pyplot as plt
import matplotlib.cm as cm
from matplotlib.colors import Normalize
import scipy
import seaborn as sns
%matplotlib inline

import librosa
import librosa.display
import IPython.display as ipd
from IPython.display import Audio

import warnings
warnings.filterwarnings('ignore')

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow import keras
from python_speech_features import mfcc
import random

from scipy import misc
import glob
from PIL import Image
from keras import layers
from keras.layers import (Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization,
                           Conv2D, AveragePooling2D, MaxPooling2D, GlobalMaxPooling2D, Dropout)
from keras.models import Model, load_model
from keras.preprocessing import image
from keras.utils import layer_utils
import pydot
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
from keras.utils.vis_utils import plot_model
from tensorflow.keras.optimizers import Adam
from keras.initializers import glorot_uniform
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from pydub import AudioSegment
import shutil
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
import random

```

Understanding Audio

In [9]:

```
# Importing 1 file
y, sr = librosa.load(f'{location}/classical/classical.00024.wav')

print('y: ', y, '\n')
print('y shape:', np.shape(y), '\n')
print('Sample Rate (KHz):', sr, '\n')

# Verify length of the audio
print('Check Len of Audio:', 661794/22050)
```

y: [-0.03775024 -0.07223511 -0.08242798 ... 0.02264404 0.00308228
-0.0105896]

y shape: (661794,)

Sample Rate (KHz): 22050

Check Len of Audio: 30.013333333333332

In []:

```
df=df.drop(labels="filename",axis=1)
```

In []:

```
audio_recording="C:/Users/Admin/Desktop/ML Project/Datasets/Train_Data/classical/classical.
data,sr=librosa.load(audio_recording)
print(type(data),type(sr))
```

In []:

```
librosa.load(audio_recording,sr)
```

Playing Audio

In []:

```
import IPython
IPython.display.Audio(data,rate=sr)
```

Waveforms

Waveforms are visual representations of sound as time on the x-axis and amplitude on the y-axis. They are great for allowing us to quickly scan the audio data and visually compare and contrast which genres might be more similar than others.

In []:

```
plt.figure(figsize=(12,4))
librosa.display.waveplot(data,color="#A300F4")
plt.title("Sound Waves in Classical 24", fontsize = 23);
plt.show()
```

Spectrogram

A spectrogram is a visual way of representing the signal loudness of a signal over time at various frequencies present in a particular waveform. Not only can one see whether there is more or less energy at, for example, 2 Hz vs 10 Hz, but one can also see how energy levels vary over time. Spectrograms are sometimes called sonographs, voiceprints, or voicegrams. When the data is represented in a 3D plot, they may be called waterfalls. In 2-dimensional arrays, the first axis is frequency while the second axis is time

In []:

```
stft=librosa.stft(data)
stft_db=librosa.amplitude_to_db(abs(stft))
plt.figure(figsize=(14,6))
librosa.display.specshow(stft,sr=sr,x_axis='time',y_axis='hz')
plt.colorbar()
```

In []:

```
stft=librosa.stft(data)
stft_db=librosa.amplitude_to_db(abs(stft))
plt.figure(figsize=(14,6))
librosa.display.specshow(stft_db,sr=sr,x_axis='time',y_axis='hz')
plt.colorbar()
```

Mel Spectrogram

In []:

```
mels = librosa.feature.melspectrogram(data)
fig = plt.figure()
canvas = FigureCanvas(fig)
p = plt.imshow(librosa.power_to_db(mels,ref=np.max))
p
```

MFCC

The Mel frequency cepstral coefficients (MFCCs) of a signal are a small set of features (usually about 10-20) that concisely describe the overall shape of a spectral envelope. It models the characteristics of the human voice. `librosa.feature.mfcc` computes MFCCs across an audio signal:

In []:

```
#using classical.00024.wav to display its MFCC
x, fs = librosa.load("C:/Users/Admin/Desktop/ML Project/Datasets/Train_Data/classical/classical.00024.wav")
librosa.display.waveplot(x, sr=fs)
mfccs = librosa.feature.mfcc(x, sr=fs)
(13, 97)
#Displaying the MFCCs:
librosa.display.specshow(mfccs, sr=fs, x_axis='time')
```

Spectral Rolloff

Spectral Rolloff is the frequency below which a specified percentage of the total spectral energy, e.g. 85%, lies. `librosa.feature.spectral_rolloff` computes the rolloff frequency for each frame in a signal.

The roll-off frequency denotes the approximate low bass and high treble limits in a frequency response curve, with all frequencies between being those a speaker will play accurately.

In []:

```
spectral_rolloff=librosa.feature.spectral_rolloff(data+0.01,sr=fs)[0]
plt.figure(figsize=(14,6))
librosa.display.waveplot(data,sr=fs,alpha=0.4,color="#2B4F72")
```

Chroma Feature

It's a useful tool for studying musical aspects whose pitches may be classified and whose tuning is close to the equal-tempered scale. Chromatic and melodic aspects of music are captured by chroma features, which are resistant to changes in timbre and instrumentation.

In []:

```
import librosa.display as lplt
chroma = librosa.feature.chroma_stft(data,sr=fs)
plt.figure(figsize=(14,6))
lplt.specshow(chroma,sr=fs,x_axis="time",y_axis="chroma",cmap="coolwarm")
plt.colorbar()
plt.title("Chroma Features")
plt.show()
```

Zero Crossing

Zero crossing is said to occur if successive samples have different algebraic signs. The rate at which zero-crossings occur is a simple measure of the frequency content of a signal. Zero-crossing rate is a measure of the number of times in a given time interval/frame that the amplitude of the speech signals passes through a value of zero.

In []:

```
start=1000
end=1200
plt.figure(figsize=(12,4))
plt.plot(data[start:end],color="#2B4F72")
plt.grid()
```

In []:

```
zero_cross_rate=librosa.zero_crossings(data[start:end],pad=False)
print("the number of zero_crossings is :", sum(zero_cross_rate))
```

In [2]:

```
location = 'C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML/Project/genres'
print(list(os.listdir(f'{location}')))
```

```
['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop',
'reggae', 'rock']
```

In [3]:

```
os.makedirs('C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML/Project/spectrograms3sec')
os.makedirs('C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms3sec/train')
os.makedirs('C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms3sec/test')
```

...

In [3]:

```
genres = 'blues classical country disco hiphop jazz metal pop reggae rock'
genres = genres.split()
```

In []:

```
for g in genres:
    path_audio = os.path.join('C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML/Project/audio')
    os.makedirs(path_audio)
    path_train = os.path.join('C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML/Project/spectrograms3sec/train')
    path_test = os.path.join('C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML/Project/spectrograms3sec/test')
    os.makedirs(path_train)
    os.makedirs(path_test)
```

In []:

```

from pydub import AudioSegment
i = 0
for g in genres:
    j=0
    print(f"{g}")
    for filename in os.listdir(os.path.join('C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML

    song = os.path.join(f'C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML/Project/genres
    j = j+1
    for w in range(0,10):
        i = i+1
        #pydub calculates in milliseconds so we multiply the values by 1000
        t1 = 3*(w)*1000
        t2 = 3*(w+1)*1000
        newAudio = AudioSegment.from_wav(song)
        new = newAudio[t1:t2]
        new.export(f'C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML/Project/audio3sec/{g}/{

```

In []:

```

for g in genres:
    j = 10
    print(g)
    for filename in os.listdir(os.path.join('C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML
    song = os.path.join(f'C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML/Project/audio3
    y, sr = librosa.load(song, duration=3)
    #print(sr)
    mels = librosa.feature.melspectrogram(y=y, sr=sr)
    fig = plt.figure()
    canvas = FigureCanvas(fig)
    p = plt.imshow(librosa.power_to_db(mels, ref=np.max))
    melspecname=(f'C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML/Project/spectrograms3se
    plt.savefig(f'{melspecname}.png', format="png")
    j = j+1

```

Splitting Dataset into 80:20

In [4]:

```

directory = "C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML/Project/spectrograms3sec1/tra
for g in genres:
    filenames = os.listdir(os.path.join(directory, f"{g}"))
    random.shuffle(filenames)
    test_files = filenames[0:20]

    for f in test_files:
        shutil.move(directory + f"{g}" + "/" + f, "C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/

```


In [5]:

```
train_dir = "C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML/Project/spectrograms3sec1/train"
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(train_dir, target_size=(288, 432), color_mode='grayscale')

validation_dir = "C:/Users/dhruv/OneDrive/Desktop/College/SEM 5/ML/Project/spectrograms3sec1/validation"
vali_datagen = ImageDataGenerator(rescale=1./255)
vali_generator = vali_datagen.flow_from_directory(validation_dir, target_size=(288, 432), color_mode='grayscale')
```

Found 800 images belonging to 10 classes.

Found 200 images belonging to 10 classes.

Building the model

All of the hidden layers are using the RELU activation function and the output layer uses the softmax function. The loss is calculated using the categorical_crossentropy function. Dropout is used to prevent overfitting. We chose the Adam optimizer because it gave us the best results after evaluating other optimizers. The model accuracy can be increased by further increasing the epochs but after a certain period, we may achieve a threshold, so the value should be determined accordingly.

In [6]:

```

def GenreModel(input_shape = (288,432,4),classes=10):
    np.random.seed(10)
    X_input = Input(input_shape)

    X = Conv2D(8,kernel_size=(3,3),strides=(1,1),kernel_initializer = glorot_uniform(seed=10))
    X = BatchNormalization(axis=3)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((2,2))(X)

    X = Conv2D(16,kernel_size=(3,3),strides = (1,1),kernel_initializer=glorot_uniform(seed=10))
    X = BatchNormalization(axis=3)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((2,2))(X)

    X = Conv2D(32,kernel_size=(3,3),strides = (1,1),kernel_initializer = glorot_uniform(seed=10))
    X = BatchNormalization(axis=3)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((2,2))(X)

    X = Conv2D(64,kernel_size=(3,3),strides=(1,1),kernel_initializer=glorot_uniform(seed=10))
    X = BatchNormalization(axis=-1)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((2,2))(X)

    X = Flatten()(X)

    X = Dropout(rate=0.3)

    X = Dense(classes, activation='softmax', name='fc' + str(classes), kernel_initializer = glorot_uniform(seed=10))

    model = Model(inputs=X_input,outputs=X,name='GenreModel')

    return model

```

In [7]:

```

import keras.backend as K
def get_f1(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val

model = GenreModel(input_shape=(288,432,4),classes=10)
opt = Adam(learning_rate=0.0005)
model.compile(optimizer = opt,loss='categorical_crossentropy',metrics=['accuracy',get_f1])

history = model.fit_generator(train_generator,epochs=70,validation_data=vali_generator)

```

Epoch 1/70

```

7/7 [=====] - 35s 5s/step - loss: 6.5165 - accur
acy: 0.1762 - get_f1: 0.1603 - val_loss: 2.3128 - val_accuracy: 0.1000 -
val_get_f1: 0.0000e+00

```

Epoch 2/70

```

7/7 [=====] - 30s 4s/step - loss: 3.5964 - accur
acy: 0.3262 - get_f1: 0.3064 - val_loss: 2.2810 - val_accuracy: 0.1000 -
val_get_f1: 0.0000e+00

```

Epoch 3/70

```

7/7 [=====] - 32s 4s/step - loss: 2.7945 - accur
acy: 0.4150 - get_f1: 0.4051 - val_loss: 2.2269 - val_accuracy: 0.1000 -
val_get_f1: 0.0000e+00

```

Epoch 4/70

```

7/7 [=====] - 32s 4s/step - loss: 1.6512 - accur
acy: 0.4212 - get_f1: 0.3931 - val_loss: 2.1074 - val_accuracy: 0.1300 -
val_get_f1: 0.0000e+00

```

Epoch 5/70

```

7/7 [=====] - 31s 4s/step - loss: 1.3990 - accur
acy: 0.5038 - get_f1: 0.4840 - val_loss: 2.0564 - val_accuracy: 0.1900 -

```

In [11]:

```
model.summary()
```

Model: "GenreModel"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 288, 432, 4)]	0
conv2d (Conv2D)	(None, 286, 430, 8)	296
batch_normalization (BatchNo	(None, 286, 430, 8)	32
activation (Activation)	(None, 286, 430, 8)	0
max_pooling2d (MaxPooling2D)	(None, 143, 215, 8)	0
conv2d_1 (Conv2D)	(None, 141, 213, 16)	1168
batch_normalization_1 (Batch	(None, 141, 213, 16)	64
activation_1 (Activation)	(None, 141, 213, 16)	0
max_pooling2d_1 (MaxPooling2	(None, 70, 106, 16)	0
conv2d_2 (Conv2D)	(None, 68, 104, 32)	4640
batch_normalization_2 (Batch	(None, 68, 104, 32)	128
activation_2 (Activation)	(None, 68, 104, 32)	0
max_pooling2d_2 (MaxPooling2	(None, 34, 52, 32)	0
conv2d_3 (Conv2D)	(None, 32, 50, 64)	18496
batch_normalization_3 (Batch	(None, 32, 50, 64)	256
activation_3 (Activation)	(None, 32, 50, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 16, 25, 64)	0
flatten (Flatten)	(None, 25600)	0
fc10 (Dense)	(None, 10)	256010
=====		
Total params: 281,090		
Trainable params: 280,850		
Non-trainable params: 240		

Model Evaluation

In [12]:

```
def plotValidate(history):
    print("Validation Accuracy", max(history.history["val_accuracy"]))
    pd.DataFrame(history.history).plot(figsize=(12,6))
    plt.show()
```

In [13]:

```
def plot_history(hist):
    plt.figure(figsize=(20,15))
    fig, axs = plt.subplots(2)
    fig.tight_layout(pad=2.0)

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

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

    plt.show()
```

In [14]:

```
history.history
```

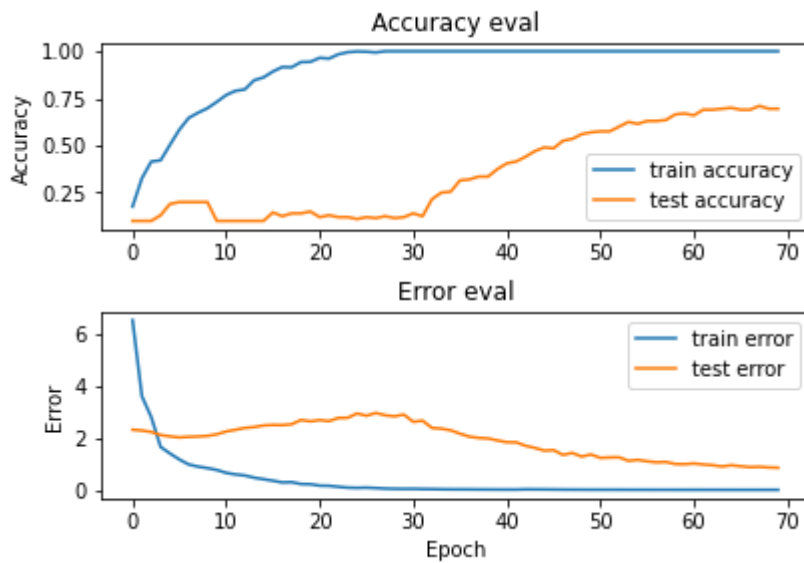
Out[14]:

```
{'loss': [6.516544342041016,
3.596397638320923,
2.7945046424865723,
1.6511744260787964,
1.398962378501892,
1.171403408050537,
0.9781506061553955,
0.8947227001190186,
0.839494526386261,
0.767581582069397,
0.6566316485404968,
0.595695972442627,
0.5559428334236145,
0.4701377749443054,
0.41194429993629456,
0.35461491346359253,
0.28889796137809753,
0.2996319830417633.]
```

In [15]:

```
plot_history(history)
```

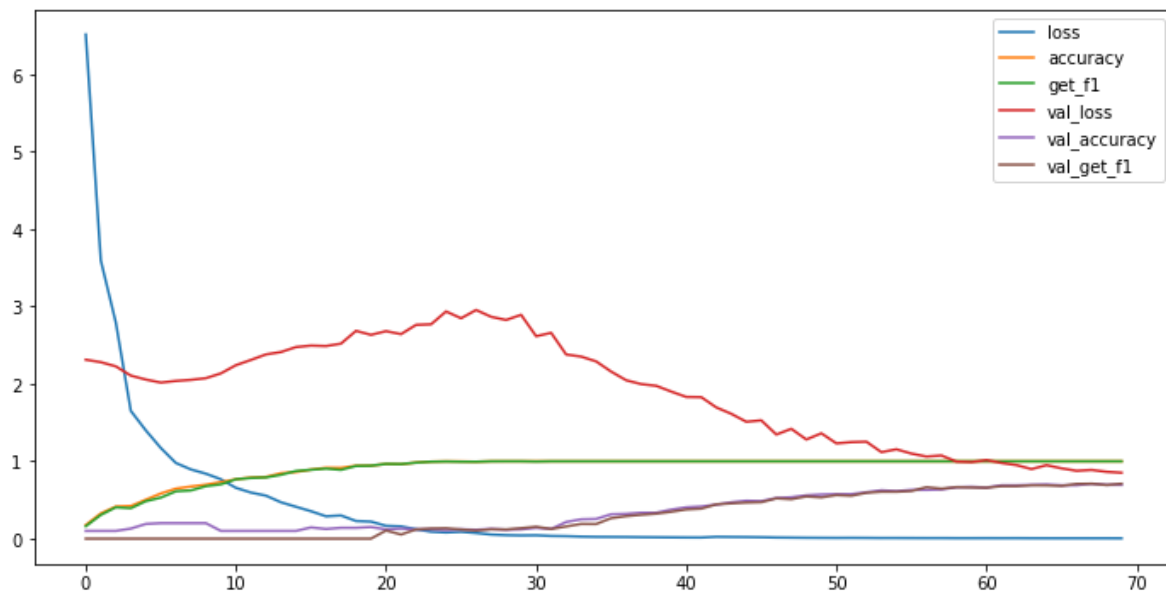
<Figure size 1440x1080 with 0 Axes>



In [16]:

```
plotValidate(history)
```

Validation Accuracy 0.7099999785423279



Prediction

In [17]:

```
class_labels = ['blues',  
'classical',  
'country',  
'disco',  
'hiphop',  
'jazz',  
'metal',  
'pop',  
'reggae',  
'rock']
```

In [18]:

```
def extract_relevant(s,t1,t2):  
    wav_file = os.path.join(f'C:/Users/dhruv/OneDrive/Desktop/',f'{s}')  
    wav = AudioSegment.from_wav(wav_file)  
    wav = wav[1000*t1:1000*t2]  
    wav.export("extracted.wav",format='wav')
```

In [19]:

```
def create_melspectrogram(wav_file):  
    y,sr = librosa.load(wav_file,duration=3)  
    mels = librosa.feature.melspectrogram(y=y,sr=sr)  
  
    fig = plt.figure()  
    canvas = FigureCanvas(fig)  
    p = plt.imshow(librosa.power_to_db(mels,ref=np.max))  
    plt.savefig('melspectrogram.png')
```

In [20]:

```
def predict(image_data,model):  
  
    #image = image_data.resize((288,432))  
    image = img_to_array(image_data)  
  
    image = np.reshape(image,(1,288,432,4))  
  
    prediction = model.predict(image/255)  
  
    prediction = prediction.reshape((10,))  
  
    class_label = np.argmax(prediction)  
  
    return class_label,prediction
```

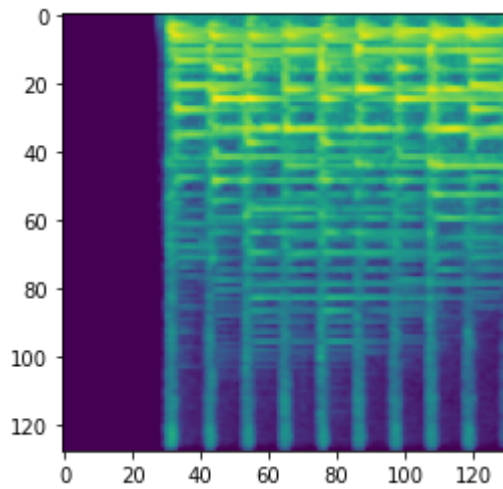
In [24]:

```
def show_output(songname):  
    extract_relevant(songname,0,100)  
    create_melspectrogram("extracted.wav")  
    image_data = load_img('melspectrogram.png',color_mode='rgba',target_size=(288,432))  
  
    class_label,prediction = predict(image_data,model)  
  
    print("## The Genre of Song is ",class_labels[class_label])  
  
    prediction = prediction.reshape((10,))  
  
    color_data = [1,2,3,4,5,6,7,8,9,10]  
    my_cmap = cm.get_cmap('jet')  
    my_norm = Normalize(vmin=0, vmax=10)  
  
    fig,ax= plt.subplots(figsize=(6,4.5))  
    ax.bar(x=class_labels,height=prediction,  
    color=my_cmap(my_norm(color_data)))  
    plt.xticks(rotation=45)  
    ax.set_title("Probability Distribution Of The Given Song Over Different Genres")  
    plt.show()
```

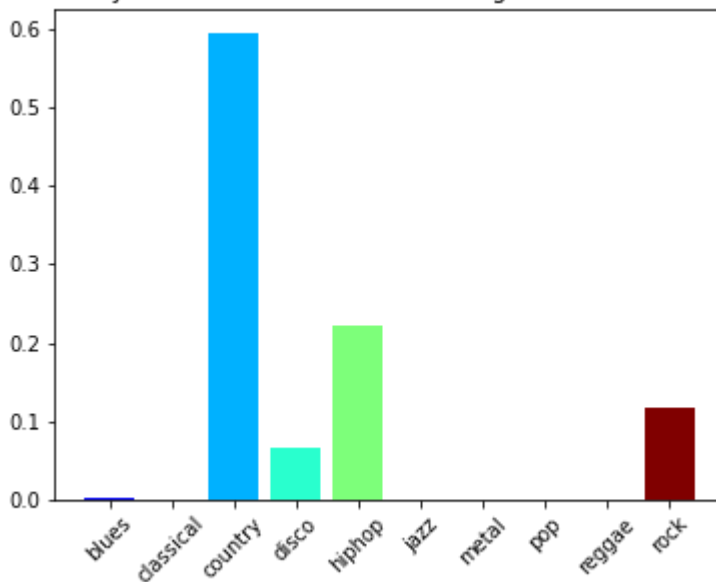

In [27]:

```
song = "Taylor Swift-Love Story.wav"  
show_output(song)
```

The Genre of Song is country



Probability Distribution Of The Given Song Over Different Genres



Installing necessary libraries

(Librosa is used for audio data)

In [1]:

```
!pip install librosa
```

In [2]:

```
!pip install python_speech_features
```

In [3]:

```
!pip install tensorflow
```

In [4]:

```
!pip install pydot
```

Requirement already satisfied: pydot in c:\users\dhruv\anaconda3\lib\site-packages (1.4.2)

Requirement already satisfied: pyparsing>=2.1.4 in c:\users\dhruv\anaconda3\lib\site-packages (from pydot) (2.4.7)

In [5]:

```
!pip install pydub
```

Requirement already satisfied: pydub in c:\users\dhruv\anaconda3\lib\site-packages (0.25.1)

Importing Libraries

In [60]:

```
import numpy as np
import pandas as pd
import os

import matplotlib.pyplot as plt
import matplotlib.cm as cm
from matplotlib.colors import Normalize
import scipy
import seaborn as sns
%matplotlib inline

import librosa
import librosa.display
import IPython.display as ipd
from IPython.display import Audio

import warnings
warnings.filterwarnings('ignore')

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow import keras
from python_speech_features import mfcc
import random

from scipy import misc
import glob
from PIL import Image
from keras import layers
from keras.layers import (Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization,
                           Conv2D, AveragePooling2D, MaxPooling2D, GlobalMaxPooling2D, Dropout)
from keras.models import Model, load_model
from keras.preprocessing import image
from keras.utils import layer_utils
import pydot
from datetime import datetime
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
from keras.utils.vis_utils import plot_model
from tensorflow.keras.optimizers import Adam
from keras.initializers import glorot_uniform
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from pydub import AudioSegment
import shutil
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
import random
```

Understanding the Audio Files

In [20]:

```
# Importing 1 file
y, sr = librosa.load(f'{location}/classical/classical.00024.wav')

print('y: ', y, '\n')
print('y shape:', np.shape(y), '\n')
print('Sample Rate (KHz):', sr, '\n')

# Verify Length of the audio
print('Check Len of Audio:', 661794/22050)
```

y: [-0.03775024 -0.07223511 -0.08242798 ... 0.02264404 0.00308228
-0.0105896]

y shape: (661794,)

Sample Rate (KHz): 22050

Check Len of Audio: 30.013333333333332

In [21]:

```
df=df.drop(labels="filename",axis=1)
```

In [61]:

```
audio_recording="C:/Users/Admin/Desktop/ML Project/Datasets/genres/classical/classical.00024.wav"
data,sr=librosa.load(audio_recording)
print(type(data),type(sr))
```

<class 'numpy.ndarray'> <class 'int'>

In [23]:

```
librosa.load(audio_recording,sr)
```

Out[23]:




```
(array([-0.03775024, -0.07223511, -0.08242798, ..., 0.02264404,
        0.00308228, -0.0105896 ], dtype=float32),
22050)
```

Playing Audio

In [62]:

```
import IPython
IPython.display.Audio(data,rate=sr)
```

Out[62]:

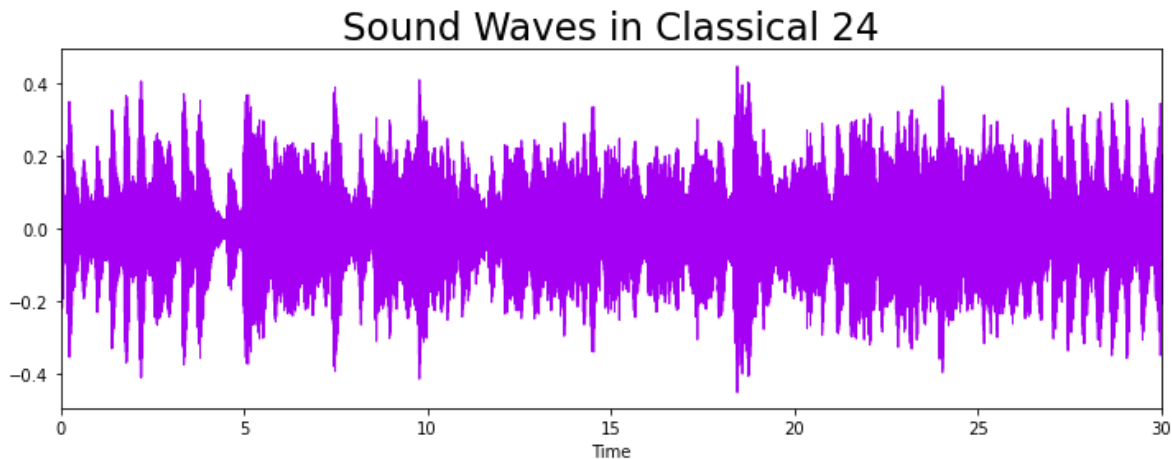
▶ 0:10 / 0:30   

Waveforms

Waveforms are visual representations of sound as time on the x-axis and amplitude on the y-axis. They are great for allowing us to quickly scan the audio data and visually compare and contrast which genres might be more similar than others.

In [25]:

```
plt.figure(figsize=(12,4))
librosa.display.waveplot(data,color="#A300F4")
plt.title("Sound Waves in Classical 24", fontsize = 23);
plt.show()
```



Spectrogram

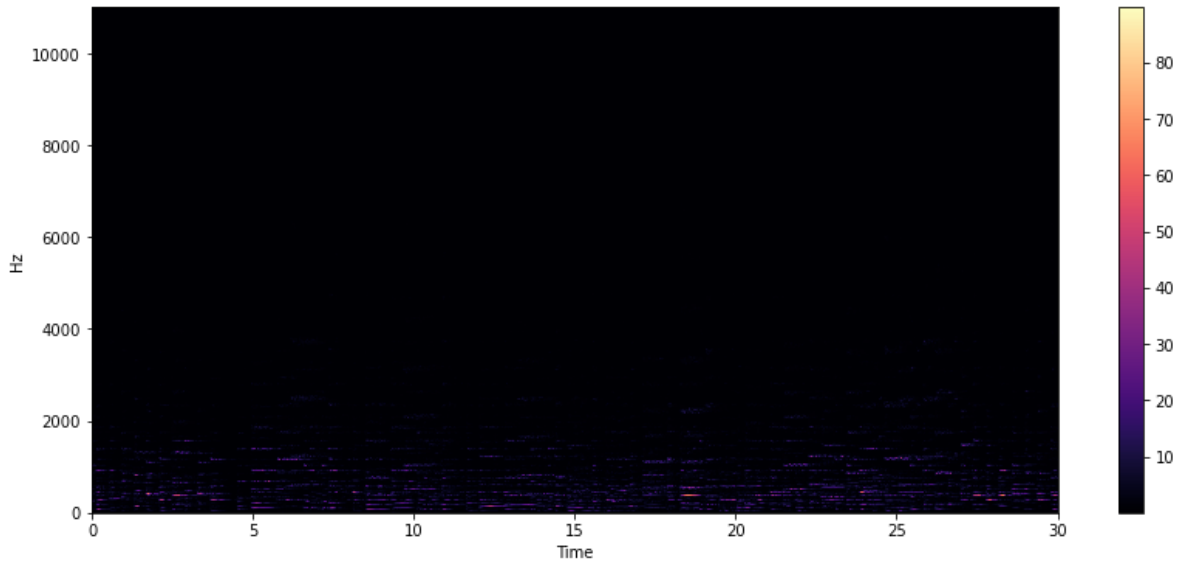
A spectrogram is a visual way of representing the signal loudness of a signal over time at various frequencies present in a particular waveform. Not only can one see whether there is more or less energy at, for example, 2 Hz vs 10 Hz, but one can also see how energy levels vary over time. Spectrograms are sometimes called sonographs, voiceprints, or voicegrams. When the data is represented in a 3D plot, they may be called waterfalls. In 2-dimensional arrays, the first axis is frequency while the second axis is time

In [27]:

```
stft=librosa.stft(data)
stft_db=librosa.amplitude_to_db(abs(stft))
plt.figure(figsize=(14,6))
librosa.display.specshow(stft,sr=sr,x_axis='time',y_axis='hz')
plt.colorbar()
```

Out[27]:

<matplotlib.colorbar.Colorbar at 0x25a8c49e790>

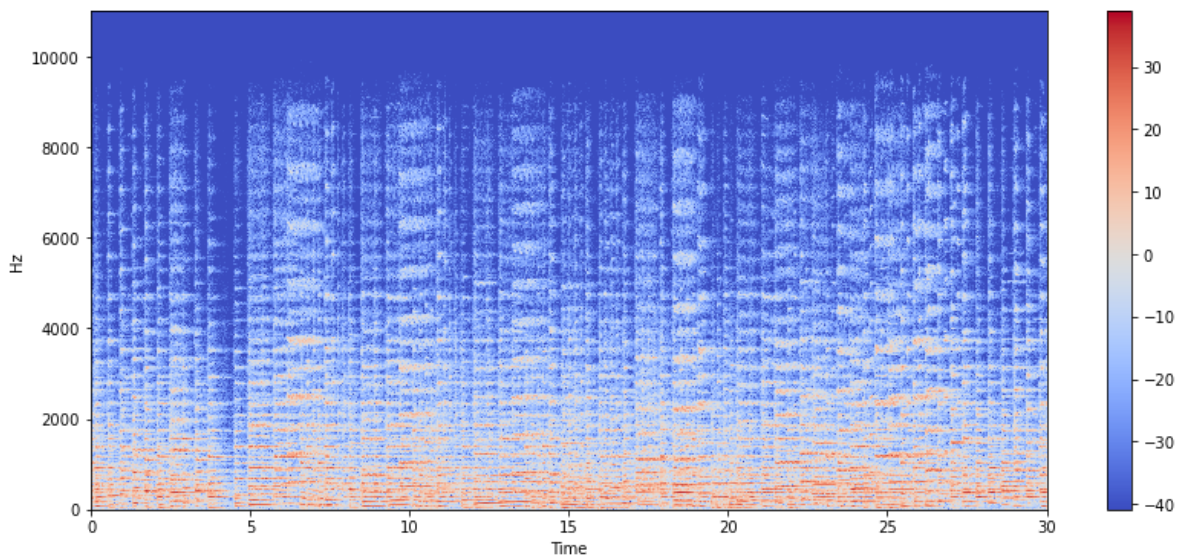


In [28]:

```
stft=librosa.stft(data)
stft_db=librosa.amplitude_to_db(abs(stft))
plt.figure(figsize=(14,6))
librosa.display.specshow(stft_db,sr=sr,x_axis='time',y_axis='hz')
plt.colorbar()
```

Out[28]:

<matplotlib.colorbar.Colorbar at 0x25a8eb79460>



Mel Spectrogram

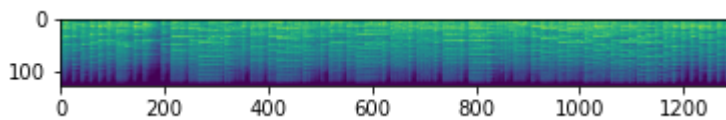
Mel spectrogram is a spectrogram that is converted to a Mel scale. The Mel scale mimics how the human ear works, with research showing humans don't perceive frequencies on a linear scale. Humans are better at detecting differences at lower frequencies than at higher frequencies.

In [63]:

```
mels = librosa.feature.melspectrogram(data)
fig = plt.figure()
canvas = FigureCanvas(fig)
p = plt.imshow(librosa.power_to_db(mels, ref=np.max))
p
```

Out[63]:

<matplotlib.image.AxesImage at 0x1fe0a594730>



MFCC

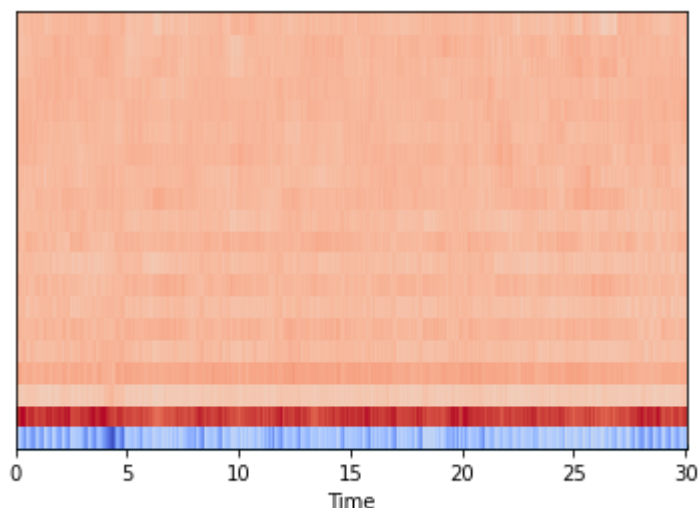
The Mel frequency cepstral coefficients (MFCCs) of a signal are a small set of features (usually about 10-20) that concisely describe the overall shape of a spectral envelope. It models the characteristics of the human voice. librosa.feature.mfcc computes MFCCs across an audio signal:

In [26]:

```
#using classical.00024.wav to display its MFCC
x, fs = librosa.load("C:/Users/Admin/Desktop/ML Project/Datasets/Train_Data/classical/class
librosa.display.waveplot(x, sr=sr)
mfccs = librosa.feature.mfcc(x, sr=fs)
(13, 97)
#Displaying the MFCCs:
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
```

Out[26]:

<matplotlib.collections.QuadMesh at 0x25a8a0d96a0>



Spectral Rolloff

Spectral Rolloff is the frequency below which a specified percentage of the total spectral energy, e.g. 85%, lies
`librosa.feature.spectral_rolloff` computes the rolloff frequency for each frame in a signal.

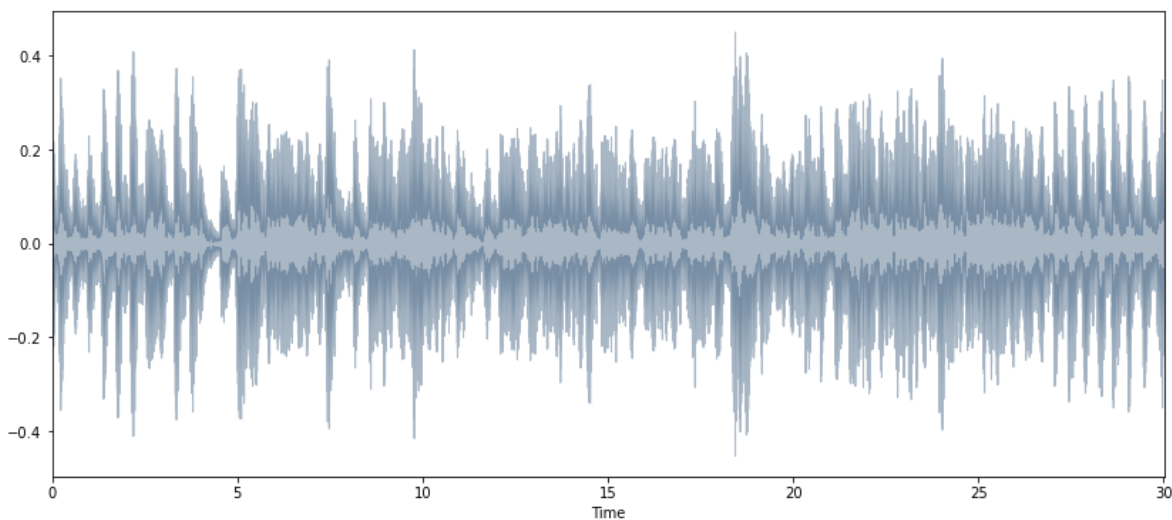
The roll-off frequency denotes the approximate low bass and high treble limits in a frequency response curve, with all frequencies between being those a speaker will play accurately

In [29]:

```
spectral_rolloff=librosa.feature.spectral_rolloff(data+0.01,sr=sr)[0]  
plt.figure(figsize=(14,6))  
librosa.display.waveplot(data,sr=sr,alpha=0.4,color="#2B4F72")
```

Out[29]:

<matplotlib.collections.PolyCollection at 0x25a8ebdf520>

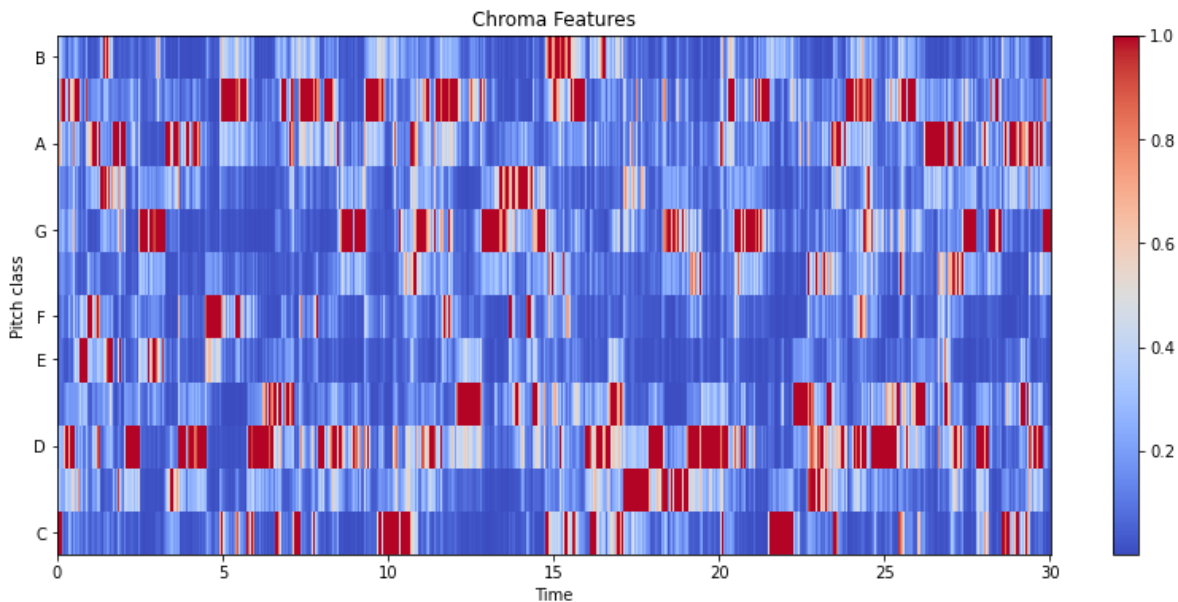


Chroma Feature

It's a useful tool for studying musical aspects whose pitches may be classified and whose tuning is close to the equal-tempered scale. Chromatic and melodic aspects of music are captured by chroma features, which are resistant to changes in timbre and instrumentation.

In [30]:

```
import librosa.display as lplt
chroma = librosa.feature.chroma_stft(data,sr=sr)
plt.figure(figsize=(14,6))
lplt.specshow(chroma,sr=sr,x_axis="time",y_axis="chroma",cmap="coolwarm")
plt.colorbar()
plt.title("Chroma Features")
plt.show()
```

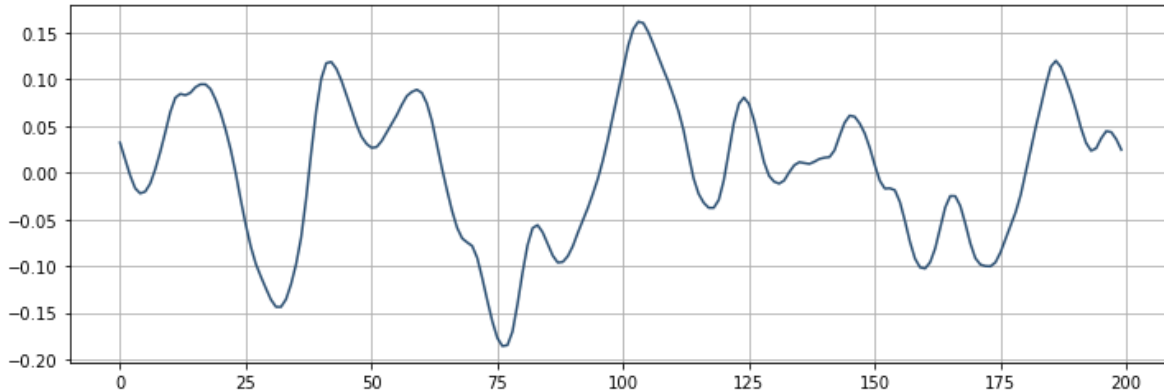


Zero Crossing

Zero crossing is said to occur if successive samples have different algebraic signs. The rate at which zero-crossings occur is a simple measure of the frequency content of a signal. Zero-crossing rate is a measure of the number of times in a given time interval/frame that the amplitude of the speech signals passes through a value of zero.

In [31]:

```
start=1000
end=1200
plt.figure(figsize=(12,4))
plt.plot(data[start:end],color="#2B4F72")
plt.grid()
```



In [32]:

```
zero_cross_rate=librosa.zero_crossings(data[start:end],pad=False)
print("the number of zero_crossings is :", sum(zero_cross_rate))
```

the number of zero_crossings is : 12

Therefore in our project, we have decided to generate Mel Spectrograms for all the audio files for the CNN Model

In [4]:

```
#Displaying the classes
location = 'C:/Users/Admin/Desktop/ML Project/Datasets/genres'
print(list(os.listdir(f'{location}')))
```

```
['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop',
'reggae', 'rock']
```

In [5]:

```
genres = 'blues classical country disco hiphop jazz metal pop reggae rock'
genres = genres.split()
```

In [6]:

```

for g in genres:
    path_audio = os.path.join('C:/Users/Admin/Desktop/ML Project/Datasets/audio30sec',f'{g}')
    os.makedirs(path_audio)
    path_train = os.path.join('C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms30sec/train',f'{g}')
    path_test = os.path.join('C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms30sec/test',f'{g}')
    os.makedirs(path_train)
    os.makedirs(path_test)

```

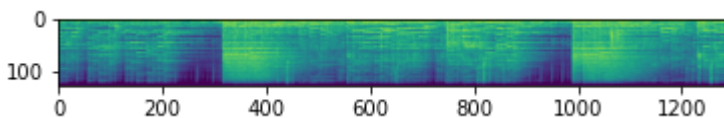
In [7]:

```

for g in genres:
    j = 0
    print(g)
    for filename in os.listdir(os.path.join('C:/Users/Admin/Desktop/ML Project/Datasets/genres',f'{g}')):
        song = os.path.join('C:/Users/Admin/Desktop/ML Project/Datasets/genres',f'{g}',f'{filename}')
        y, sr = librosa.load(song,duration=30)
        #print(sr)
        mels = librosa.feature.melspectrogram(y=y,sr=sr)
        fig = plt.figure()
        canvas = FigureCanvas(fig)
        p = plt.imshow(librosa.power_to_db(mels,ref=np.max))
        melspecname=(f'C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms30sec/train/{g}/{filename}.png')
        plt.savefig(f'{melspecname}.png', format="png")
        j = j+1

```

blues
classical
country
disco
hiphop
jazz
metal
pop
reggae
rock



In [10]:

```

directory = "C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms30sec/train/"
for g in genres:
    filenames = os.listdir(os.path.join(directory,f'{g}'))
    random.shuffle(filenames)
    test_files = filenames[0:10]

    for f in test_files:
        shutil.move(directory + f'{g}' + "/" + f, "C:/Users/Admin/Desktop/ML Project/Datasets/spe

```

In [25]:

```
train_dir = "C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms30sec/train/"
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(train_dir,target_size=(288,432),color_m

validation_dir = "C:/Users/Admin/Desktop/ML Project/Datasets/spectrograms30sec/test/"
vali_datagen = ImageDataGenerator(rescale=1./255)
vali_generator = vali_datagen.flow_from_directory(validation_dir,target_size=(288,432),colo
```

Found 900 images belonging to 10 classes.
Found 100 images belonging to 10 classes.

In [50]:

```
class_labels = ['blues',
                'classical',
                'country',
                'disco',
                'hiphop',
                'jazz',
                'metal',
                'pop',
                'reggae',
                'rock']
```

In [23]:

```
def GenreModel(input_shape = (288,432,4),classes=10):
    np.random.seed(10)
    X_input = Input(input_shape)

    X = Conv2D(8,kernel_size=(3,3),strides=(1,1),kernel_initializer = glorot_uniform(seed=10))
    X = BatchNormalization(axis=3)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((2,2))(X)

    X = Conv2D(16,kernel_size=(3,3),strides = (1,1),kernel_initializer=glorot_uniform(seed=10))
    X = BatchNormalization(axis=3)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((2,2))(X)

    X = Conv2D(32,kernel_size=(3,3),strides = (1,1),kernel_initializer = glorot_uniform(seed=10))
    X = BatchNormalization(axis=3)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((2,2))(X)

    X = Conv2D(64,kernel_size=(3,3),strides=(1,1),kernel_initializer=glorot_uniform(seed=10))
    X = BatchNormalization(axis=-1)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((2,2))(X)

    X = Flatten()(X)

    X = Dropout(rate=0.3)

    X = Dense(classes, activation='softmax', name='fc' + str(classes), kernel_initializer = glorot_uniform(seed=10))

    model = Model(inputs=X_input,outputs=X,name='GenreModel')

    return model
```

In [20]:

```
def plotValidate(history):
    print("Validation Accuracy",max(history.history["val_accuracy"]))
    pd.DataFrame(history.history).plot(figsize=(12,6))
    plt.show()
```

In [21]:

```
def plot_history(hist):
    plt.figure(figsize=(20,15))
    fig, axs = plt.subplots(2)
    fig.tight_layout(pad=2.0)

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

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

    plt.show()
```

In [38]:

```

import keras.backend as K
def get_f1(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val

model = GenreModel(input_shape=(288,432,4),classes=10)
opt = Adam(learning_rate=0.0005)
model.compile(optimizer = opt,loss='categorical_crossentropy',metrics=['accuracy',get_f1])

history = model.fit_generator(train_generator,epochs=70,validation_data=vali_generator)

```

Epoch 60/70

8/8 [=====] - 51s 7s/step - loss: 0.8658 - accuracy: 0.7211 - get_f1: 0.7155 - val_loss: 2.2351 - val_accuracy: 0.3800 - val_get_f1: 0.4072

Epoch 61/70

8/8 [=====] - 52s 6s/step - loss: 0.9737 - accuracy: 0.6844 - get_f1: 0.5942 - val_loss: 2.8498 - val_accuracy: 0.3600 - val_get_f1: 0.3407

Epoch 62/70

8/8 [=====] - 51s 6s/step - loss: 1.1552 - accuracy: 0.6267 - get_f1: 0.6242 - val_loss: 3.3192 - val_accuracy: 0.3200 - val_get_f1: 0.3297

Epoch 63/70

8/8 [=====] - 52s 6s/step - loss: 0.7833 - accuracy: 0.7178 - get_f1: 0.6990 - val_loss: 2.9933 - val_accuracy: 0.3000 - val_get_f1: 0.3069

Epoch 64/70

8/8 [=====] - 51s 6s/step - loss: 0.8872 - accuracy: 0.7022 - get_f1: 0.6548 - val_loss: 2.0291 - val_accuracy: 0.3800 - val_get_f1: 0.3676

In [39]:

```
model.summary()
```

Model: "GenreModel"

Layer (type)	Output Shape	Param #
=====		
input_4 (InputLayer)	[(None, 288, 432, 4)]	0
conv2d_12 (Conv2D)	(None, 286, 430, 8)	296
batch_normalization_12 (Batch Normalization)	(None, 286, 430, 8)	32
activation_12 (Activation)	(None, 286, 430, 8)	0
max_pooling2d_12 (MaxPooling2D)	(None, 143, 215, 8)	0
conv2d_13 (Conv2D)	(None, 141, 213, 16)	1168
batch_normalization_13 (Batch Normalization)	(None, 141, 213, 16)	64
activation_13 (Activation)	(None, 141, 213, 16)	0
max_pooling2d_13 (MaxPooling2D)	(None, 70, 106, 16)	0
conv2d_14 (Conv2D)	(None, 68, 104, 32)	4640
batch_normalization_14 (Batch Normalization)	(None, 68, 104, 32)	128
activation_14 (Activation)	(None, 68, 104, 32)	0
max_pooling2d_14 (MaxPooling2D)	(None, 34, 52, 32)	0
conv2d_15 (Conv2D)	(None, 32, 50, 64)	18496
batch_normalization_15 (Batch Normalization)	(None, 32, 50, 64)	256
activation_15 (Activation)	(None, 32, 50, 64)	0
max_pooling2d_15 (MaxPooling2D)	(None, 16, 25, 64)	0
flatten_3 (Flatten)	(None, 25600)	0
fc10 (Dense)	(None, 10)	256010
=====		
Total params: 281,090		
Trainable params: 280,850		
Non-trainable params: 240		

In [40]:

```
history.history
```

Out[40]:

```
{'loss': [6.6283955574035645,  
4.4528961181640625,  
3.172081470489502,  
2.4042489528656006,  
2.1584043502807617,  
2.6329410076141357,  
2.217747211456299,  
2.4022533893585205,  
2.098011016845703,  
1.992236852645874,  
2.2798714637756348,  
2.054011344909668,  
1.951963186264038,  
2.1113550662994385,  
2.1906986236572266,  
1.8053557872772217,  
1.640149712562561,  
1.8705106973648071.]
```

In [41]:

```
plot_history(history)
```

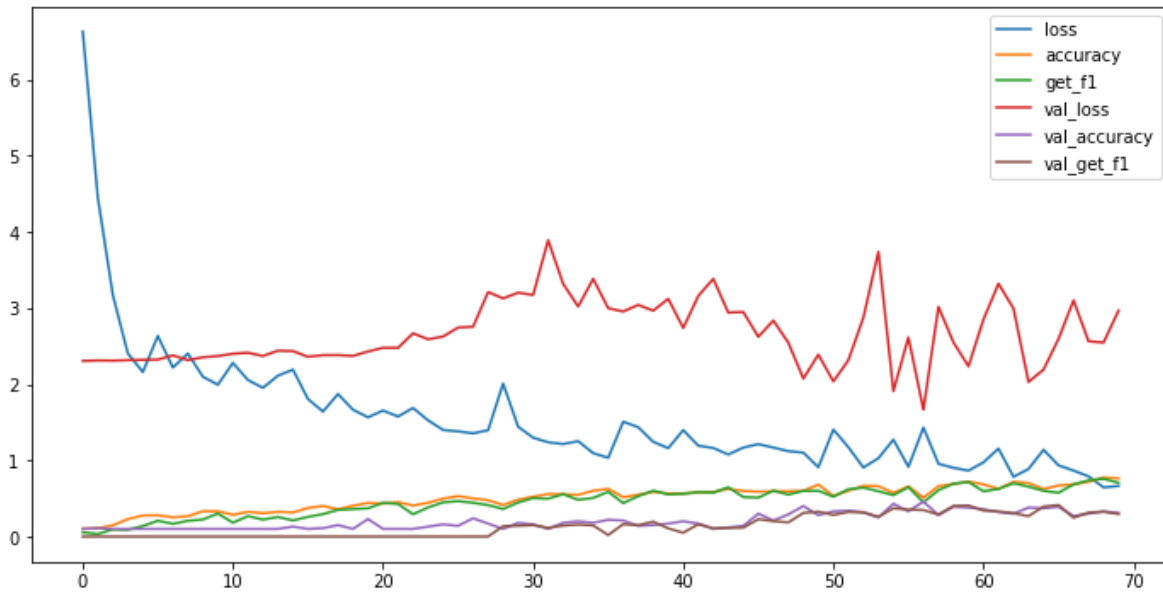
<Figure size 1440x1080 with 0 Axes>



In [42]:

```
plotValidate(history)
```

Validation Accuracy 0.4600000834465027



In [43]:

```
parameters = model(X)
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-43-ee5deab7fafc> in <module>
----> 1 parameters = model(X)
```

NameError: name 'X' is not defined

In [44]:

```
def convert_mp3_to_wav(music_file):
    sound = AudioSegment.from_mp3(music_file)
    sound.export("music_file.wav", format="wav")
```

In [45]:

```
def extract_relevant(s,t1,t2):  
    wav_file = os.path.join(f'C:/Users/Admin/Desktop/ML Project/Datasets/sample/',f'{s}')  
    wav = AudioSegment.from_wav(wav_file)  
    wav = wav[1000*t1:1000*t2]  
    wav.export("extracted.wav",format='wav')
```

In [46]:

```
def create_melspectrogram(wav_file):  
    y,sr = librosa.load(wav_file,duration=3)  
    mels = librosa.feature.melspectrogram(y=y,sr=sr)  
  
    fig = plt.figure()  
    canvas = FigureCanvas(fig)  
    p = plt.imshow(librosa.power_to_db(mels,ref=np.max))  
    plt.savefig('melspectrogram.png')
```

In [47]:

```
def predict(image_data,model):  
  
    #image = image_data.resize((288,432))  
    image = img_to_array(image_data)  
  
    image = np.reshape(image,(1,288,432,4))  
  
    prediction = model.predict(image/255)  
  
    prediction = prediction.reshape((10,))  
  
    class_label = np.argmax(prediction)  
  
    return class_label,prediction
```

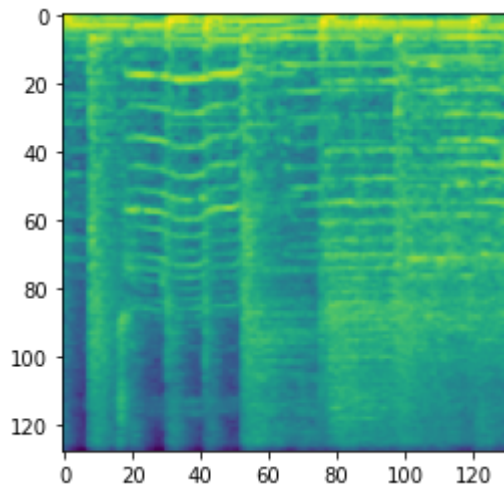
In [48]:

```
def show_output(songname):  
    extract_relevant(songname,40,50)  
    create_melspectrogram("extracted.wav")  
    image_data = load_img('melspectrogram.png',color_mode='rgba',target_size=(288,432))  
  
    class_label,prediction = predict(image_data,model)  
  
    print("## The Genre of Song is ",class_labels[class_label])  
  
    prediction = prediction.reshape((10,))  
  
    color_data = [1,2,3,4,5,6,7,8,9,10]  
    my_cmap = cm.get_cmap('jet')  
    my_norm = Normalize(vmin=0, vmax=10)  
  
    fig,ax= plt.subplots(figsize=(6,4.5))  
    ax.bar(x=class_labels,height=prediction,  
    color=my_cmap(my_norm(color_data)))  
    plt.xticks(rotation=45)  
    ax.set_title("Probability Distribution Of The Given Song Over Different Genres")  
    plt.show()
```

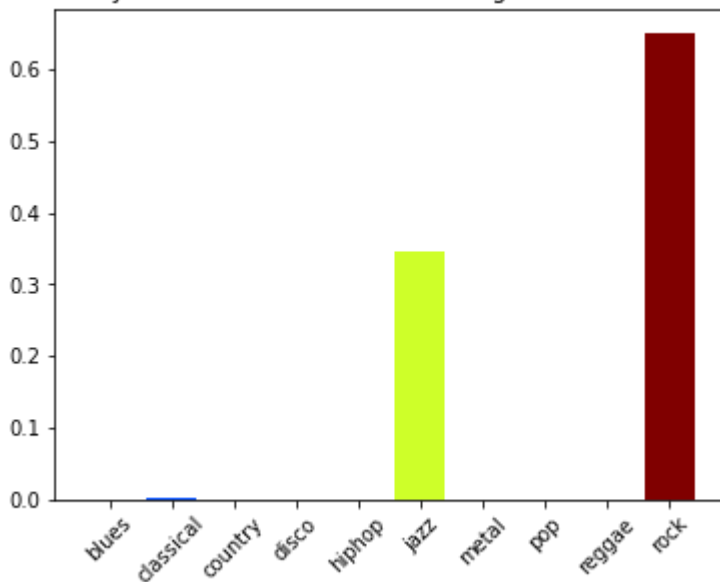
In [51]:

```
song = "NirvanaSmellsLikeTeenSpirit.wav"  
show_output(song)
```

The Genre of Song is rock



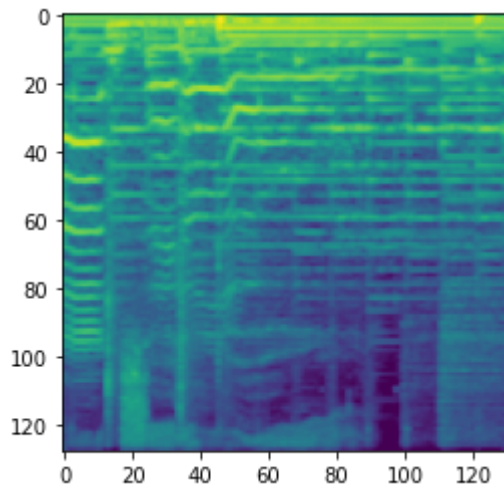
Probability Distribution Of The Given Song Over Different Genres



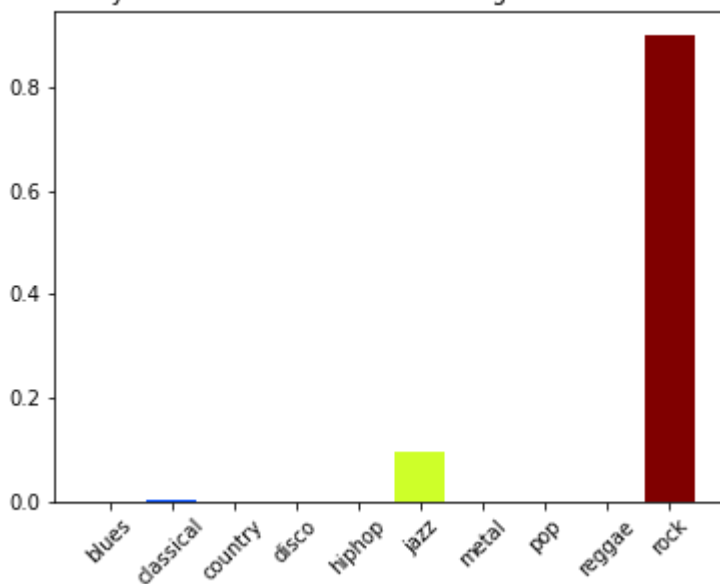
In [64]:

```
song = "TaylorSwiftLoveStory.wav"  
show_output(song)
```

The Genre of Song is rock



Probability Distribution Of The Given Song Over Different Genres



In []:

