FOREWORD

We are submitting the **EHB328E Machine Learning for Signal Processing** course term project report entitled as **Music Genre Classification**. We hereby confirm that we have realized all stages of the course project work by ourselves and we have abided by the ethical rules with respect to academic and professional integrity.

January 2021

Aurel ELMAZAJ - 040170928
Berkan YAPICI - 040160604
Enes DEMİRAĞ - 040160027

TABLE OF CONTENTS

Page

ABBREVIATIONS

CNN	**:** Concolutional Neural Network
DCT	**:** Discrite Cosine Transform
FMA	**:** Free Music Archive
FFT	**:** Fast Fourier Transform
KNN	**:** K-Nearest Neighbors
MFCC	**:** Mel-Frequency Cepstral Coefficients
MLP	**:** Multi Layer Perceptron
PCA	**:** Principal Component Analysis
RBF	**:** Radial Basis Function
STFT	**:** Short-Time Fourier transform
SVM	**:** Support Vector Machine
ZCR	**:** Zero Crossing Rate

LIST OF FIGURES

Page

Music Genre Classification

SUMMARY

Music has different features such as the instruments used, rhythm, pitch, voices, lyrics and many more. Different musics that are similar to each other in terms of these features creates a common genre. However, there is no sharp boundary between different genres. While some genres of music are easy to distinguish from each other, some are very similar to each other.

The need to classify music according to the relevant genre is an area where streaming services and music player applications work on. For example, a music streaming service can find more specific music suggestions by determining which genre the music users listen to.

There are many different studies in this field. The aim of our study is to examine the different machine learning algorithms used in the music genre classification problem in depth and compare their results by implementing them.

1. INTRODUCTION

First, we conducted independent research during the first weeks of the topic selection phase. Later, we shared our thoughts and decided on "Music Genre Classification" as the project subject. Our initial thinking is to decide on a dataset and process dimension reduction and feature extraction will be designed in the first step. Then the training process will be started with various machine learning algorithms. It has been decided to use Python programming language, librosa library [1], scikit-learn library [2] and tensorflow library [3] to be used in this project. Librosa library will help us to preprocess the data. Scikit-learn is chosen to use classical machine learning algorithms such as Support Vector Machine (SVM) and Nearest Neighbors (KNN). On the other hand, tensorflow will be used for neural network models like Multi Layer Perceptron (MLP) and Convolutional Neural Networks (CNN).

1.1 Dataset

Two datasets that are widely used in this field have been examined. The first of these is FMA [4] which stands for free music archive. This dataset contains 106574 pieces of tracks, consisting of 161 different genres in total. It contains metadata information as well as audio data. The other dataset is GTZAN [5]. This dataset is smaller in size compared to the first one and contains 1000 audio samples in total, 100 tracks from 10 different genres. Each audio sample is mono channel, 16 bit .wav formated file with a sampling rate of 22050 Hz. It has been decided to use GTZAN dataset to be used in the project at least in the beginning. Because a cleaner and smaller dataset with less number of classes will be efficient to achieve better results in a limited time.

For comparison, in this kind of genre classification problem, the human accuracy average is 70% [6]. Previous works in this problem were achieved great results. It is possible to obtain an accuracy up to 60% with a simple K'th Nearest Neighbor algorithm with hand-crafted features [7]. And better results have been obtained in this problem by using a Multiple Layer Support Vector Machine [8].

2. DATA PREPROCESSING

As previously mentioned, we used GTZAN [5] dataset in our project. Instead of feeding audio tracks directly into the classification models, we had to introduce some data preprocessing in the first place. Data preprocessing refers to a principal step of data mining process which converts the raw data into a suitable, compatible and understandable way for machine learning applications. As a result, by reducing the dimension of the data and extracting essential features, better accuracy could be achieved. So, first of all, we tried to visualize our dataset by making use of both, waveform representation and most importantly, spectrogram representation. The latter was largely studied by us and therefore, different types of spectrograms were obtained. These spectrograms were saved as images and fed into CNN models in order to compare the effectiveness of deep learning versus traditional machine learning algorithms for this type of audio classification implementation. The next step was to employ feature extraction on the dataset. By using the needed libraries, we have calculated 25 features in total, 20 of them were mel-frequency cepstral coefficents (MFCCs) [9]. Other important features include: spectral bandwidth, spectral roll-off, spectral centroid [10], zero crossing rate and chromagram [11]. These features were computed for each audio of all genres. Lastly, by using pandas library, the mentioned 25 features were saved to a csv file with respective genre as a label. The csv file was later used as an input to implement several classification models.

2.1 Data Visualization

In this sections, different representations of the audio data will be expained.

2.1.1 Audio Waveforms

We started the visualization by firstly implementing a basic waveform representation to have a simple idea of how our sound data looks like. Waveform is a graph which shows the shape and form of the sound signal and explains how the amplitude varies

over time. Hence, the horizontal axis corresponds to time and the vertical axis to amplitude. Using librosa and sklearn libraries, with a sampling rate of 22050 Hz, we managed to plot the waveforms. An example of the used code is given below and its output for a random song of genre hip hop can be seen in figure 2.1.

```
import librosa, IPython
import librosa.display
import sklearn
import matplotlib.pyplot as plt

sample_file = 'hiphop.00005.wav'
signal , sr = librosa.load(sample_file , sr = 22050)
plt.figure(figsize=(15,5))
librosa.display.waveplot(signal , sr = sr)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title("HipHop music signal")
plt.show()
```

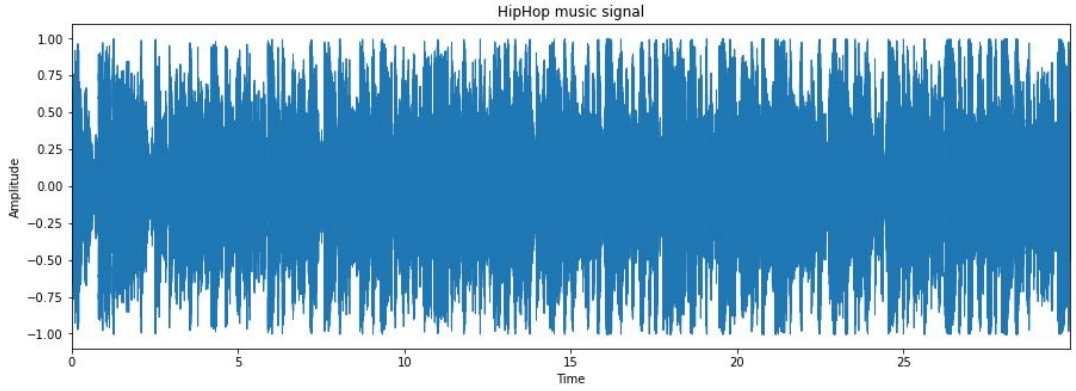

Figure 2.1 : Waveform of a Hip-Hop music

2.1.2 Spectrograms

Spectrogram is another important representation of audio signals. It helps us to visualize the signal strength at different frequencies over time. The horizontal axis corresponds to time, vertical axis to frequency and the intensity of color to amplitude. Unlike waveforms, spectrograms also give frequency information, making them useful for speech and music analysis. A spectrogram is obtained by applying short-time Fourier transforms on signal waveforms. STFT is a digital signal processing technique based on a sequence of fast Fourier transforms (FFTs) [12] which are calculated on overlapping windowed segments. Therefore, it can be seen as several FFTs placed on

top of each-other. These windows of the signal are usually allowed to overlap in time approximately 25-50%. Again, by using stft tool of librosa library, we can easily plot the spectrograms of our data like shown in figure 2.2.

```
X = librosa.stft(music)
X_db = librosa.amplitude_to_db(abs(X))
librosa.display.specshow(X_db, sr=sr, x_axis='time', y_axis='hz')
```

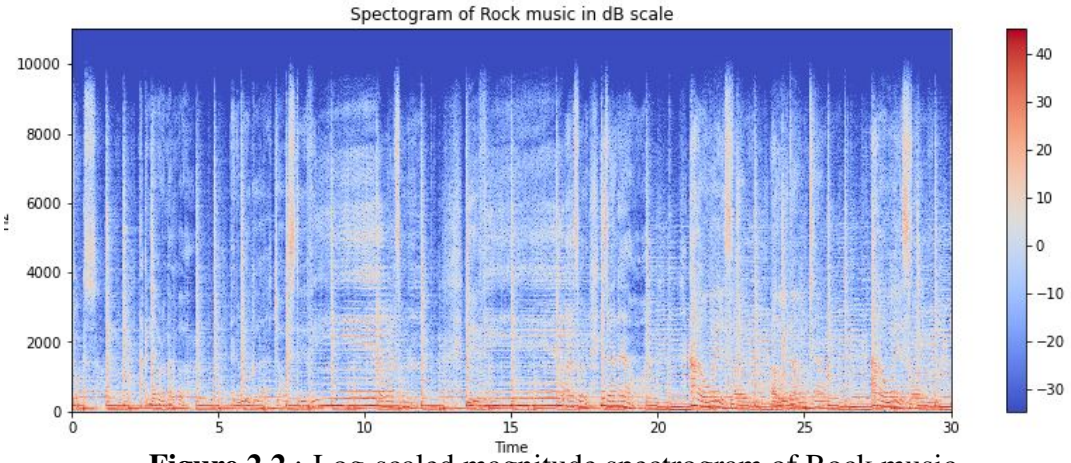

Figure 2.2 : Log-scaled magnitude spectrogram of Rock music

However, we have decided to use mel-spectrograms for our models. Mel-spectrogram is just a spectrogram whose frequencies are converted into mel-scale [13]. This is quite important because it is well-known that people does not perceive frequencies on a linear scale. Instead, we sense the frequencies logarithmically, meaning that lower frequencies are dominant for human ears. In a mel scale, equal distances have same perceptual representation. To obtain such spectrograms, we first execute STFT to waveforms and convert amplitude to dB scale. Lastly and most importantly, the frequencies should be transformed to Mel-scale. This is realized by firstly deciding the number of mel bands, then building the mel triansgular filters and apply them to the spectrograms. All these mel-spectrograms were saved as images and later applied to CNN models. Using again the librosa library, mel-spectrogram can be utilized with these 2 lines of code, and output will look like in figure 2.3.

```
mel_spec = librosa.feature.melspectrogram(signal, sr=sr)
mel_spec_db = librosa.amplitude_to_db(mel_spec, ref=np.max)
```

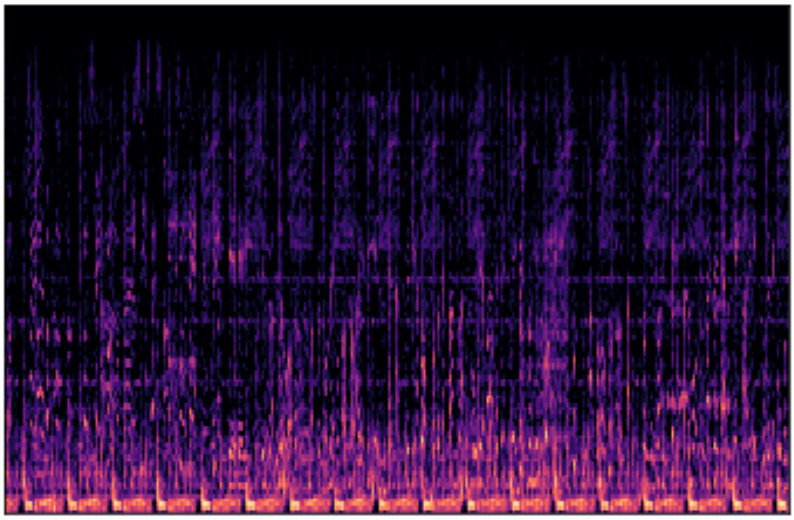

Figure 2.3 : Mel spectrogram of Hip-Hop song

2.1.3 PCA Visualization

In addition to the previous methods, we used Principal Component Analysis [14] to visualize the GTZAN dataset. PCA is an important technique used for different applications such as: data visualization, data compression, noise reduction, data classification etc. In our case, by using the following code, we reduced the data to 2 dimensions and plotted that in a 2D graph in figure 2.4.

```
# Using 2 PCA components
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(dataset)
```

2.2 Feature Extraction

In this section, the features we gathered from audios will be explained.

2.2.1 Mel-Frequecny Cepstral Coefficents

Mel-frequency cepstral coefficents or MFCCs [9] are a set of features which best describe the shape and spectrum of a a mel-spectrogram in a compressed way. In our csv file, we are using 20 MFCCs for all songs in each particular genre. In theory, these coefficents are obtained by firstly taking STFT of the signals. Then, triangular

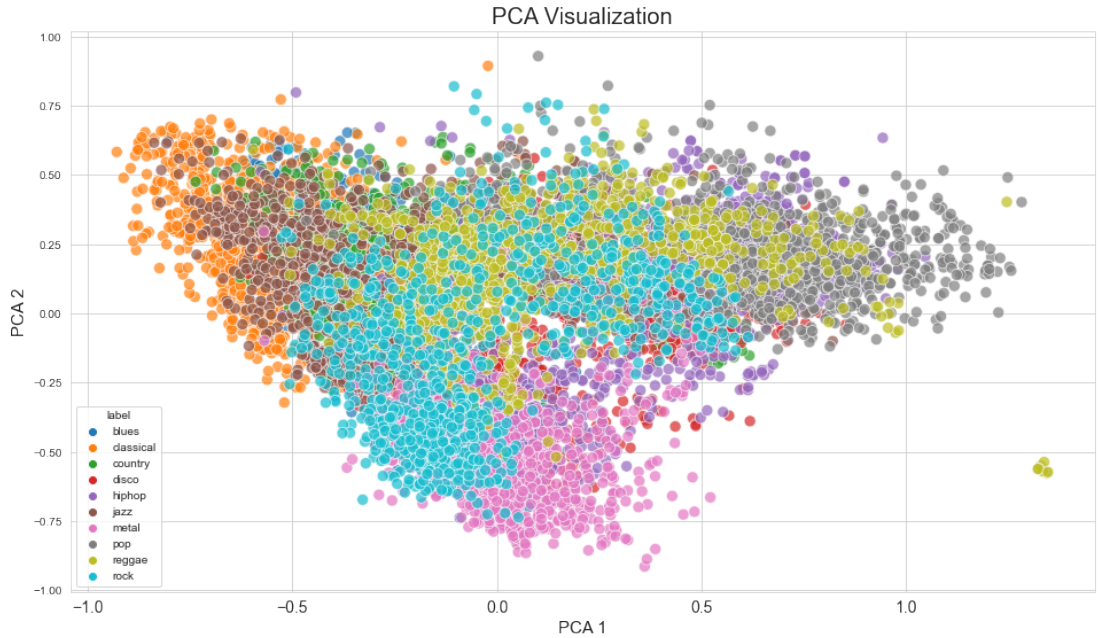

Figure 2.4 : PCA visualization using 2 components

filters are applied to transform the spectrum into a mel-scale. Furthermore, the power magnitude is converted in a logarithmic scale and DCT is used to obtain the coefficents. Basically, MFFCs are the magnitudes of mel-spectrograms. In practice, they can be obtained using librosa library as shown below:

```
mfccs = librosa.feature.mfcc(signal, sr=sr)
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
```

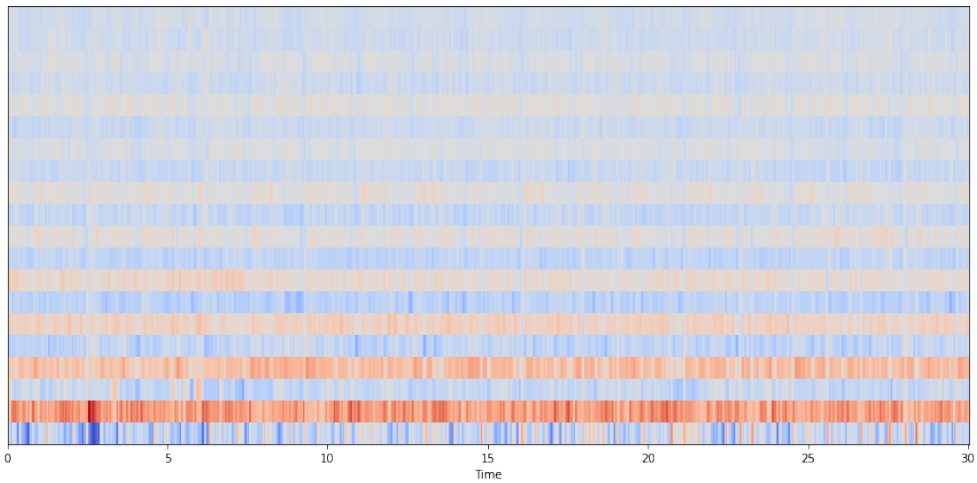

Figure 2.5 : MFFCs plot versus time

2.2.2 Chroma Frequencies

Chroma frequencies [11] are important features for music audio analysis. Similar to spectrograms, chromagrams are another way of representing the sound signals based on waveforms of 12 different pitch classes. Chroma values are denoted as C, C, D, D, E, F, F, G, G, A, A, B respectively. These features are extracted in a way that they capture harmonic and melodic characteristics of music. By using STFT and other techniques, the values give a harmonic description. Example shown below is the chromagram of the previous hip-hop sound with its respective implemented code.

```
hop_length = 512
chromagram = librosa.feature.chroma_stft(signal,
                                         sr=sr,
                                         hop_length=hop_length)
```

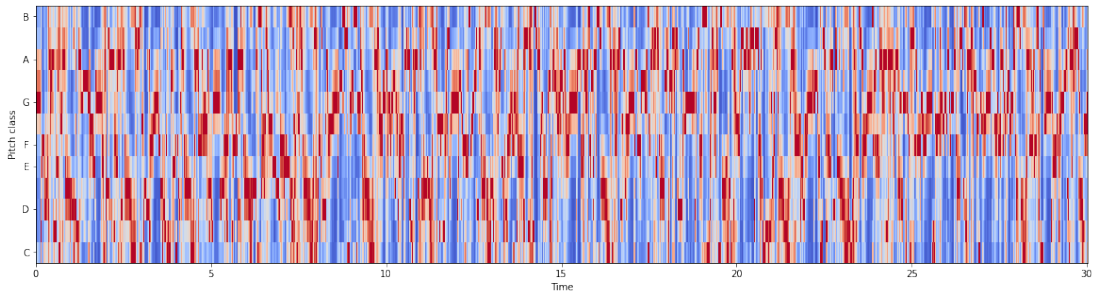

Figure 2.6 : Chromagram of Hip-Hop song

2.2.3 Spectral Centroid

Spectral centroid [10] is another feature of a spectrum which tells us the frequency in which the energy of the spectrum is concentrated, or in other words indicates where the center of mass of a particular sound is located. It is widely used in music processing applications. Theoretically, spectral centroid value can be obtained using the equation 2.1.

$$Spectral\ Centroid = \frac{\sum_{k=0}^{N} k * w * A_k}{\sum_{k=0}^{N} A_k} \tag{2.1}$$

In this formula, w is the width of each spectral bin k and A is the corresponding amplitude. Using again librosa library we evaluated the spectral centroid of each song with the following one-line code.

```
spectral_centroids = librosa.feature.spectral_centroid(signal,sr=sr)
```

2.2.4 Spectral Bandwidth

Spectral bandwidth is a parameter related to the width of the band of light at one-half the peak maximum. It can be calculated using the code shown below.

```
spectral_bandwidth_2 = librosa.feature.spectral_bandwidth(signal, sr=sr)
```

2.2.5 Zero Crossing Rate

Zero crossing rate is a measurement used to describe the rate of sign-changes of a signal during a given interval. In other words, it tells how fast the signal goes from positive to zero to negative and vice-versa. Analytically, ZCR of a signal x[n] can be obtained using the equation 2.2.

$$Z = \frac{1}{2 * W_l} \sum_{n=1}^{W_l} |sgn[x_i(n-1)]| \tag{2.2}$$

In python, we can calculate via the code:

```
zero_crossings = librosa.zero_crossings(signal[n0:n1])
```

2.2.6 Spectral Rolloff

Lastly, spectral rolloff is a feature which describes the frequency below which a given amount of spectral energy is located. It is usually used to give a quantitive measurement of the shape of a signal. In our datasets, spectrall rolloff represents the frequency at which higher frequencies goes to zero. It is computed using the code.

```
spectral_rolloff = librosa.feature.spectral_rolloff(signal+0.01, sr=sr)
```

3. TRADITIONAL METHODS

In our study, before using artificial neural network models, we tried traditional machine learning algorithms that do not require training. After the feature extraction phase, we designed simple but successful classifier models with widely used machine learning algorithms such as KNN and SVM using the new csv data we obtained from audio files.

3.1 K-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm is a supervised machine learning algorithm that is simple and easy to implement. It is a method that can be used in the solution of both classification and regression problems. This algorithm assumes that the data points with the same labels are in close proximity in feature space. An example of 2D dataset can be seen in figure 3.1. Data points with the same labels are located at close distance to each other. KNN decides which class a new input data belongs to by calculating its distance from other known data points in the dataset.

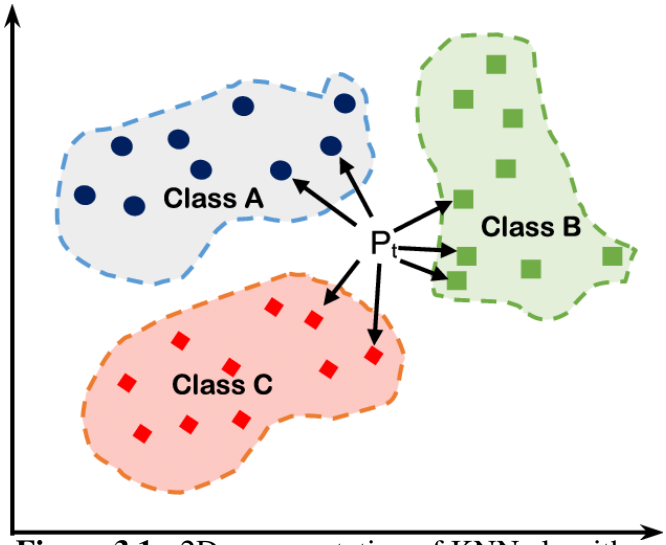

Figure 3.1 : 2D representation of KNN algorithm

To explain the KNN algorithm step by step, first, when a new input data comes, the distance to each existing data is calculated. This distance could be, for example, an euclidean distance which is the square root of the sum of every distances of dimensions.

Then, the closest k samples are taken from the collection sorted according to these distances. Here, an odd number is usually chosen for the k parameter when working on classification problems. Afterwards, among the selected data points, which class has the most members, the new input is classified as the same class.

In our implementation, first of all, we performed the feature extraction stage by reading all the audio files in the music genre dataset with the help of librosa library. Then, this new prepared dataset is divided into train and test sub-sets to measure the accuracy of the KNN model. We fit several different KNN models by using train sets with different k values to achieve the best result. By examining the accuracy scores obtained with the test set, we determined the most appropriate value for our dataset as k = 9.

Our KNN classifier model python code shown below. With this aproach, we achieved 62% accuracy.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier

dataset = load_analytic_data("dataset.csv")

# Encoding the labels
genres = dataset.iloc[:, -1] # Last column
encoder = LabelEncoder()
labels = encoder.fit_transform(genres)

# Scaling the features
scaler = StandardScaler() # MinMaxScaler() can be also used
features = scaler.fit_transform(np.array(dataset.iloc[:, :-1], dtype=float))

# Dividing dataset into training and testing sets 80to20 split
x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.2)

# Create knn model
model = KNeighborsClassifier(n_neighbors=9, weights="distance")

# Training
model.fit(x_train, y_train)

# Testing
accuracy = model.score(x_test, y_test)
print(accuracy)
```

3.2 Support Vector Machine

Support Vector Machine (SVM) is another popular supervised machine learning algorithm. The purpose of this method is to specify boundaries in n dimensional space that will divide classes from each other in the best way. In this algorithm, extreme data samples called support vectors are determined and a hyperplane is created to differentiate classes with the help of these support vectors. Any new input data is classified according to its location to these hyperplanes. The simplest SVM model is obtained using a linear hyperplane.

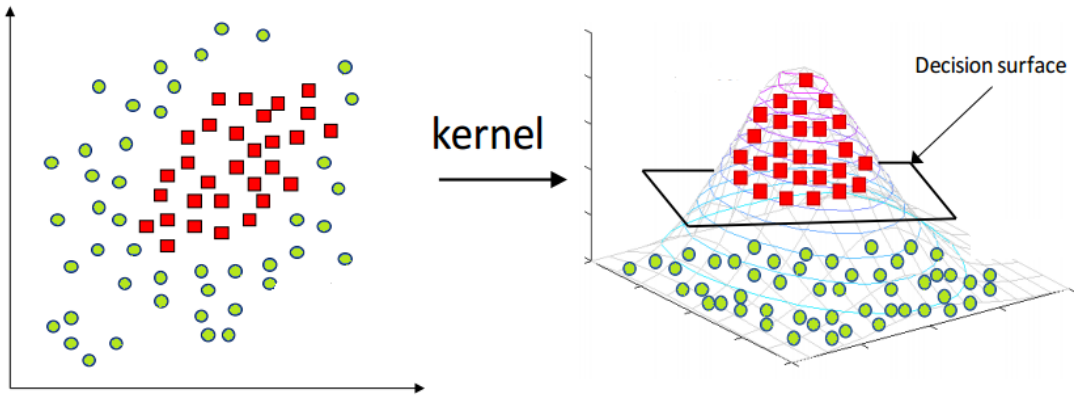

Figure 3.2 : SVM Kernel Trick

In the case of a linearly non-seperable dataset, using a linear hyperplane will not give sufficient results. To solve this problem, a method called kernel trick can be used. Kernel is a non-linear function that transforms a low dimensional input space into a higher dimensional space. It is possible to make a linearly non-seperable problem seperable with the kernel trick. An example of SVM kernel trick can be seen in figure 3.2. For example, the gaussian radial basis kernel function is given in equation 3.1.

$$k(x_i, x_j) = \exp(-\gamma \left\| x_i - x_j \right\|^2), \qquad \gamma = 1/2\sigma^2 \tag{3.1}$$

In our case, we used radial basis function (RBF) as a kernel function. Using the same features like KNN, we utilized SVM classifier model and got better accuracy. Our achieved accuracy with SVM model is 67%. Our SVM classifier model python code shown below.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC

dataset = load_analytic_data("dataset.csv")

# Encoding the labels
genres = dataset.iloc[:, -1] # Last column
encoder = LabelEncoder()
labels = encoder.fit_transform(genres)

# Scaling the features
scaler = StandardScaler() # MinMaxScaler() can be also used
features = scaler.fit_transform(np.array(dataset.iloc[:, :-1], dtype=float))

# Dividing dataset into training and testing sets 80to20 split
x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.2)

# Create knn model
model = SVC(kernel='rbf')

# Training
model.fit(x_train, y_train)

# Testing
accuracy = model.score(x_test, y_test)
print(accuracy)
```

4. DEEP LEARNING METHODS

4.1 Convolutional Neural Network

The first deep learning method applied is convolutional neural networks. CNN is used on the spectrogram images created. The main idea behind the convolutional neural networks is, extracting the main features using convolution kernels. Kernels slide through the values and extract different characteristics of images. After the convolution, activation function is used to add nonlinearity and obtain values in a more specific range. Then, pooling layers are used to shrink the input image in order to reduce the computational load and number of parameters.The pooling layer's kernel takes the maximum value of the given kernel, average pooling can also a choice for the pooling layer, and assign it. After repeating this process a couple of times, output of the last pooling layer flattened, reshaped to a 1 by m tensor, then feeded to a regular feed forward network consisting fully connected layers followed by activation layers and finally attached to the final layer outputs the predictions.

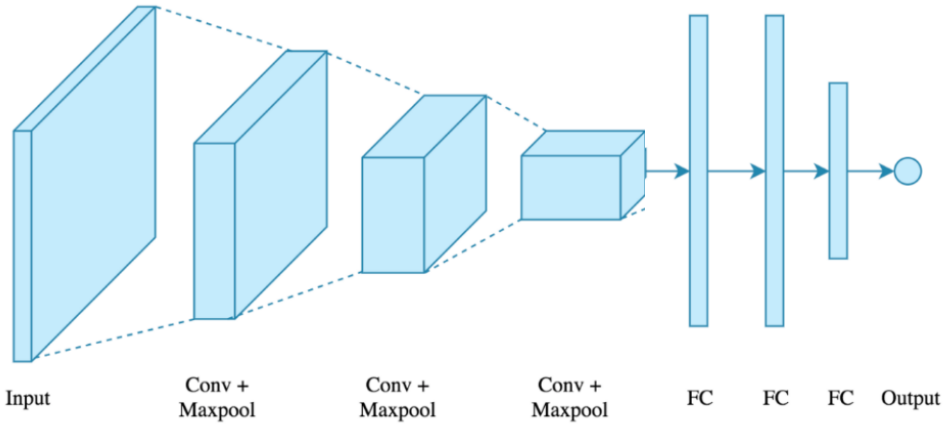

Figure 4.1 : Example CNN layer structure

The first step is loading the input, but we can't load all the images to RAM at once because the size of the input is too big, that is why we used a keras function to load data to model in batches. 800 files used for training and 200 files used for validation.

```
batch_size = 32
img_height = 256
img_width = 256
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    shuffle = True,
    image_size = (img_height, img_width),
    label_mode = 'categorical',
    batch_size=batch_size)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    shuffle = True,
    image_size = (img_height, img_width),
    label_mode = 'categorical',
    batch_size=batch_size)
```

The second step is creating CNN. After some literature observations and trying a wide range of networks best results come from a CNN architecture consisting of 3 base blocks followed by 2 dense layers. After experimenting with the model, additional dropout layers are added to prevent the CNN from memorizing and overfitting. The model starts with a normalization layer. Then base blocks extracts the main characteristic of the images. After base blocks, Flatten() function creates a 1 by m tensor, then the flattened tensor goes into the fully connected layers. Then the final layer has a softmax activation layer to assign probabilities to each category. Softmax function can be seen in equation 4.1. The callback method is used to save the best parameters of CNN in the training process. The tensorboard is used to visualize the training and validation accuracies of the training.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \; for\; i = 1,2,...,K \qquad z = (z_1, z_2, ..., z_K) \in \mathbb{R}^K \tag{4.1}$$

```
model = Sequential([
    experimental.preprocessing.Rescaling(1./255, input_shape=shape),

    Conv2D(32, (3,3), padding='valid'),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),

    Dropout(0.30),
    Conv2D(32, (3,3), padding='valid'),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),

    Dropout(0.30),
    Conv2D(32, (3,3), padding='valid'),
    Activation('relu'),
    MaxPooling2D(pool_size=(2, 2)),

    Flatten(),
    Dropout(0.3),
    Dense(64, activation='relu'),

    Dropout(0.30),
    Dense(32, activation='relu'),

    Dense(10, activation='softmax')])

my_callbacks = [
    EarlyStopping(monitor='val_loss', patience=20),
    ModelCheckpoint(filepath='./'+NAME,
                    monitor='val_loss',
                    save_best_only=True),
    TensorBoard(log_dir='logs/{}'.format(NAME))]

history = model.fit(Train_ds,
                    validation_data=val_ds,
                           batch_size=batch_size,
                          epochs=600,
                        callbacks = [my_callbacks])

test_loss, test_acc = model.evaluate(val_ds, verbose=2)
- loss: 1.4891 - acc: 0.6181
```

After the tuning steps we only reached 61.8% accuracy on the validation data. The poor results can be caused by the small number of training data. Additional data augmentation techniques can improve the results.

4.2 Multi Layer Perceptron

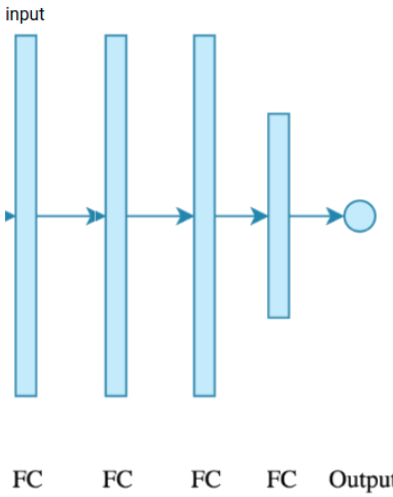

Figure 4.2 : Example MLP layer structure

The second deep learning method applied is multilayer perceptron. The multilayer perceptron is used on the mel-frequency-cepstral coefficients (MFFCs). First step is the preprocessing step, data scaled so that the mean of columns becomes zero, standard deviation becomes 1. Then data is splitted into train and test sets. Then the train data goes into MLP architecture which consists of 5 layers. 1 input layer, 3 fully connected dense layers and one output layer. The structure has a total of 28,746 parameters.

```
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.4),

    Dense(128, activation='relu'),
    Dropout(0.3),

    Dense(64, activation='relu'),
    Dropout(0.3),

    Dense(10, activation='softmax')])
```

The model achieved a test accuracy of 63.9%. The model is very accurate finding the classical music and metal. On the other hand it has a hard time classifying disco, rock, hip hop and reggae. Confusion matrix can be seen in figure 4.3

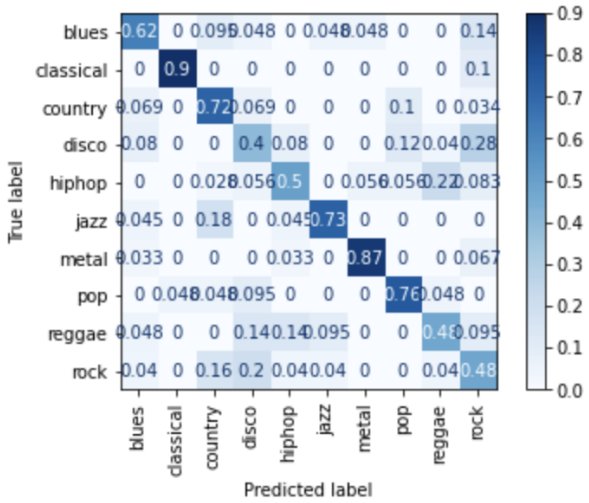

Figure 4.3 : Confusion Matrix

5. CONCLUSION

In this report, we explained the decisions we made during our studies. First, we determined a dataset for our study, and we conducted studies on dimension reduction and feature extraction on this dataset. All the analytical features we extracted from audios mentioned in previous chapters. Then we achieved results using traditional machine learning methods such as KNN and SVM. After that, we designed more complex models using artificial neural network structures such as CNN and MLP. In these modern approaches, other than previously extracted features, we also used spectrograms as inputs. Spectrograms and how we managed to create them was mentioned in previous chapters. Finally, after achieving sufficient results with different approaches, we compared their accuracies.

5.1 Results

Additional methods are implemented to see the accuracies of different methods different than mentioned in the report. As seen from the table most accurate are the ones which use support vector classifiers. The traditional methods are performed better than the deep learning methods that can be related to the size of the data. Deep learning methods need enormous amounts of data, with more data we can reach better scores with CNN and MLP methods. Also traditional methods can be improved with applying some additional feature extraction techniques to the dataset.

MODEL	ACCURACY	BALANCED ACCURACY	F1 SCORE
SVC	0.66	0.67	0.65
LGBMClassifier	0.66	0.67	0.65
CalibratedClassifierCV	0.64	0.67	0.63
NuSVC	0.65	0.67	0.64
MLP	0.64	0.64	0.64
LinearSVC	0.64	0.67	0.64
ExtraTreesClassifier	0.62	0.64	0.62
KNeighborsClassifier	0.62	0.63	0.61
CNN	0.618	0.59	-
XGBClassifier	0.61	0.62	0.61
RandomForestClassifier	0.60	0.62	0.60

Table 5.1 : Accuracies of different algorithms

5.2 Possible Applications of this Project

With this study, using these classification methods, a recommender system for a music streaming service can be designed. Also its possible to create an application to classify and categorize any size of disorganized music archives into genres.

REFERENCES

[1] B. McFee, V. Lostanlen, A. Metsai, M. McVicar, S. Balke, C. Thomé, C. Raffel, F. Zalkow, A. Malek, Dana, K. Lee, O. Nieto, J. Mason, D. Ellis, E. Battenberg, S. Seyfarth, R. Yamamoto, K. Choi, viktorandreevichmorozov, J. Moore, R. Bittner, S. Hidaka, Z. Wei, nullmightybofo, D. Hereñú, F.-R. Stöter, P. Friesch, A. Weiss, M. Vollrath, and T. Kim, "librosa/librosa: 0.8.0," Jul. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3955228

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[4] M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson, "FMA: A Dataset For Music Analysis," *arXiv:1612.01840 [cs]*, Sep. 2017, arXiv: 1612.01840. [Online]. Available: http://arxiv.org/abs/1612.01840

[5] G. Tzanetakis, G. Essl, and P. Cook, "Automatic musical genre classification of audio signals," 2001. [Online]. Available: http://ismir2001.ismir.net/pdf/tzanetakis.pdf

[6] M. Dong, "Convolutional Neural Network Achieves Human-level Accuracy in Music Genre Classification," *arXiv:1802.09697 [cs, eess]*, Feb. 2018, arXiv: 1802.09697. [Online]. Available: http://arxiv.org/abs/1802.09697

[7] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002.

[8] Changsheng Xu, N. C. Maddage, Xi Shao, Fang Cao, and Qi Tian, "Musical genre classification using support vector machines," in *2003 IEEE*

International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03)., vol. 5, 2003, pp. V–429.

[9] S. Molau, M. Pitz, R. Schluter, and H. Ney, "Computing mel-frequency cepstral coefficients on the power spectrum," in *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, vol. 1, 2001, pp. 73–76 vol.1.

[10] J. M. Grey and J. W. Gordon, "Perceptual effects of spectral modifications on musical timbres," *The Journal of the Acoustical Society of America*, vol. 63, no. 5, pp. 1493–1500, 1978. [Online]. Available: https://doi.org/10.1121/1.381843

[11] A. Shah, M. Kattel, A. Nepal, and D. Shrestha, "Chroma feature extraction," 01 2019.

[12] E. O. Brigham and R. E. Morrow, "The fast fourier transform," *IEEE Spectrum*, vol. 4, no. 12, pp. 63–70, 1967.

[13] M. Sahidullah and G. Saha, "Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition," *Speech Communication*, vol. 54, no. 4, pp. 543 – 565, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167639311001622

[14] I. Jolliffe, "Principal Component Analysis," in *International Encyclopedia of Statistical Science*, M. Lovric, Ed. Berlin, Heidelberg: Springer, 2011, pp. 1094–1096. [Online]. Available: https://doi.org/10.1007/978-3-642-04898-2_455