National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Jonathan Mendes
Student ID: x18179584

School of Computing
National College of Ireland

Supervisor: Dr Rashmi Gupta

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Jonathan Mendes
Student ID:	x18179584
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr Rashmi Gupta
Submission Due Date:	28/09/2020
Project Title:	Configuration Manual
Word Count:	895
Page Count:	21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	28th September 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Jonathan Mendes
x18179584

1 Introduction

The configuration manual consists of the steps and the procedure used to implement the research titled, 'Deep Learning Techniques for Music Genre Classification and Building a Music Recommendation System'. Detailed information on the system specifications, environment, data source, data extraction, and cleaning, pre-processing, and data modeling techniques are mentioned in this document. The various libraries and packages used in the code are listed.

2 Environment Specifications

The data extraction, cleaning, pre-processing and exploratory data analysis are carried out on a 64-bit Windows Operating System. The hardware and software configurations are listed below:

2.1 Hardware

- Processor: AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx, 2100 Mhz
- RAM: 8 GB
- System Type: Windows 10 OS, 64-bit
- GPU: NVIDIA GeForce GTX 1650, 4GB
- Storage: 512 GB SSD

2.2 Software

- Microsoft Excel for Office 365 MSO (16.0.12527.20880) 64-bit: The metadata track file of the audio files are stored in the CSV (comma separated values) format
- Anaconda Distribution-Jupyter Notebook: The data extraction, cleaning, pre-processing and exploratory data analysis is carried out using python (version 3.7.3) on Jupyter Notebook (version 6.0.0)
- Google Colaboratory Pro: The neural network classification models, recommendation model, and evaluations are carried out on Google Colaboratory Pro because of the higher GPU requirement. A Google Colaboratory Pro account needs to be got.

Following packages and libraries have been used in this research:

- import pandas as pd - library used for manipulations and data calculations.
- import numpy as np - library used for carrying out mathematical functions
- import cv2 - library designed to resolve computer vision issues
- import io - to provide stream handling with Python interfaces
- import librosa - package for audio and music analysis
- import matplotlib.pyplot - a collection of functions to make matplotlib function like MATLAB
- import os - to move folders, remove folders, create folders and change the working directory
- from PIL import Image- a library that adds support for saving, manipulating and opening various image file formats
- import re - library for regular expression
- from tqdm import tqdm - to enable a smart progress meter for loops
- import warnings - module to warn about changes in library or language features
- import logging - module for logging
- from collections import defaultdict, Counter - containers to store data collections
- import sys - to access the functions and others defined inside the module utilizing the module name
- import tensorflow - a library that uses data flow graphs to develop models
- from tensorflow.keras.models import Model, load_model
- from tensorflow.keras import optimizers
- import time - to handle various time operations
- import random - to use random() functions from this package
- from sklearn.model_selection import train_test_split - library to shuffle and split the data
- from tensorflow.keras.callbacks import ModelCheckpoint
- from tensorflow.keras.utils import plot_model, to_categorical
- from tensorflow.keras.models import Sequential
- from tensorflow.keras import initializers

- from tensorflow.keras.layers import AveragePooling2D, Conv2D, Conv3D, BatchNormalization, Flatten, Dense, Dropout, Input, Reshape, Permute, LSTM, TimeDistributed, Bidirectional
- from keras import backend - library to provide building blocks of a high-level for developing deep learning systems

3 Data Collection

The data is collected from the Free Music Archive (FMA) dataset present at `https://github.com/mdeff/fma` (Defferrard et al.; 2016). The fma_small zip file is used containing the 8000 audio tracks of 30 seconds each in the mp3 format. There are 8 balanced genres of rock, pop, hip-hop, instrumental, electronic, international, folk. The genre details for the audio tracks is obtained from the track metadata file in the CSV format.

4 Exploratory Data Analysis

The track metadata in the CSV format and the 8000 audio tracks are used for the exploratory data analysis. Figure 1 displays the track metadata in a data frame in Python.

Out[8]:

	album										...	track			
	comments	date_created	date_released	engineer	favorites	id	information	listens	producer	tags	...	information	interest	language_code	licens
track_id															
2	0	2008-11-26 01:44:45	2009-01-05 00:00:00	NaN	4	1		6073	NaN	[]	...	NaN	4656	en	NonC Sha
3	0	2008-11-26 01:44:45	2009-01-05 00:00:00	NaN	4	1		6073	NaN	[]	...	NaN	1470	en	NonC Sha
5	0	2008-11-26 01:44:45	2009-01-05 00:00:00	NaN	4	1		6073	NaN	[]	...	NaN	1933	en	NonC Sha
10	0	2008-11-26 01:45:08	2008-02-06 00:00:00	NaN	4	6	NaN	47632	NaN	[]	...	NaN	54881	en	NonC No
20	0	2008-11-26 01:45:05	2009-01-06 00:00:00	NaN	2	4	"spiritual songs" from Nicky Cook	2710	NaN	[]	...	NaN	978	en	NonC No
26	0	2008-11-26 01:45:05	2009-01-06 00:00:00	NaN	2	4	"spiritual songs" from Nicky Cook	2710	NaN	[]	...	NaN	1060	en	NonC No
30	0	2008-11-26 01:45:05	2009-01-06 00:00:00	NaN	2	4	"spiritual songs" from Nicky Cook	2710	NaN	[]	...	NaN	718	en	NonC No
46	0	2008-11-26 01:45:05	2009-01-06 00:00:00	NaN	2	4	"spiritual songs" from Nicky Cook	2710	NaN	[]	...	NaN	252	en	NonC No
48	0	2008-11-26 01:45:05	2009-01-06 00:00:00	NaN	2	4	"spiritual songs" from Nicky Cook	2710	NaN	[]	...	NaN	247	en	NonC No
134	0	2008-11-26 01:44:45	2009-01-05 00:00:00	NaN	4	1		6073	NaN	[]	...	NaN	1126	en	NonC Sha

Figure 1: Track metadata

The unwanted columns are removed from the dataframe and the updated data frame is as seen in Figure 2.

	set	track	track_id
	subset	genre_top	
track_id			
2	small	Hip-Hop	2
5	small	Hip-Hop	5
10	small	Pop	10
140	small	Folk	140
141	small	Folk	141
148	small	Experimental	148
182	small	Rock	182
190	small	Folk	190
193	small	Folk	193
194	small	Folk	194

Figure 2: Required Columns

The genre distribution of the dataset is balanced as seen in Figure 3. The 8 genres of rock, pop, hip-hop, electronic, international, folk, instrumental and experimental have 1000 songs each.

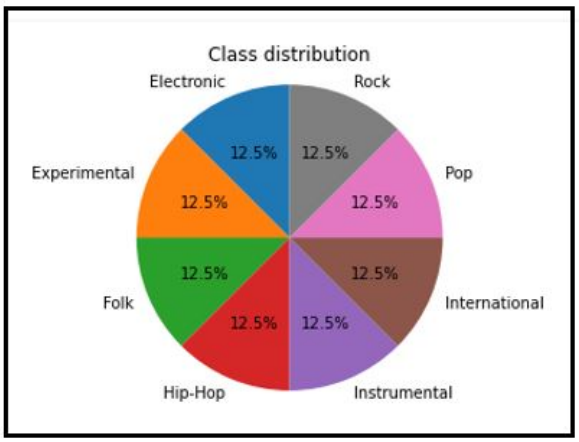

Figure 3: Balanced Dataset

The Mel-spectrograms for the 8 genres are seen in Figure 4 using the Librosa Python library.

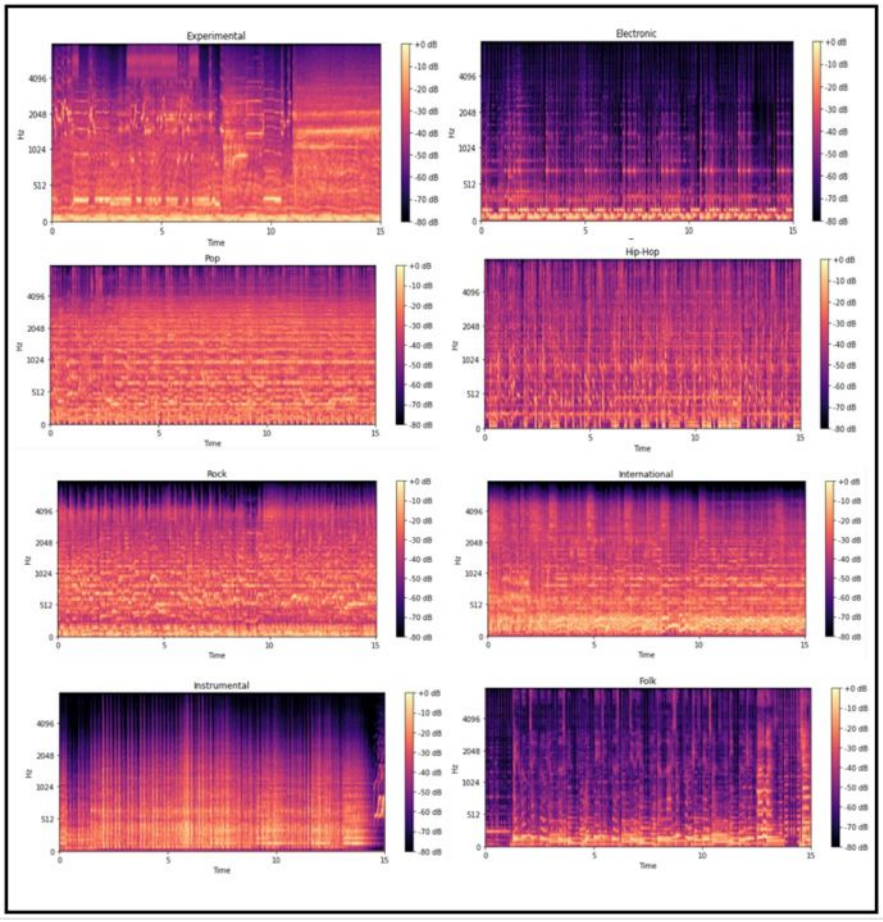

Figure 4: Mel-spectrograms for the 8 genres

Below is the Python code snippet to generate the Mel-spectrogram

```
def generate_spectrogram(trackid, genre):
    filename = fetch_audio_path(audio_dir, trackid)
    y, sr = librosa.load(filename)
    print(len(y),sr)
    spectro = librosa.feature.melspectrogram(y = y, sr = sr,
        n_fft = 2048, hop_length = 1024)
    spectro = librosa.power_to_db(spectro, ref = np.max)
    print(spectro.shape, genre)
    plt.figure(figsize = (10, 4))
    librosa.display.specshow(spectro, y_axis = 'mel',
        fmax = 8000, x_axis = 'time')
    plt.colorbar(format = '%+2.0f dB')
    plt.title(str(genre))
    plt.show()

for idx, row in grouped_df.iterrows():
    trackid = int(row['track_id'])
    genre = row[('track', 'genre_top')]
    generate_spectrogram(trackid, genre)
```

5 Preprocessing

Below is the code snippet for the Mel-spectrogram feature extraction and cleaning process.

```
def mp3_to_n_slice_np(file_path):
    """
    Function to convert .mp3 file to 10 slice mel spectrogram
    """
    y, sr = librosa.load(file_path)

    #Load mp3 file
    melspectrogram_array = librosa.feature.melspectrogram(y=y,
        sr=sr, n_mels=128, fmax=8000)

    #raise to db values
    mel = librosa.power_to_db(melspectrogram_array)

    #Covert mel values to image and slice into 10 parts
    plt.rcParams['figure.dpi'] = 100
    fig_size = plt.rcParams["figure.figsize"]
    fig_size[0] = float(mel.shape[1]) / float(100)
    fig_size[1] = float(mel.shape[0]) / float(100)
    plt.rcParams["figure.figsize"] = fig_size
    plt.axis('off')
    plt.axes([0., 0., 1., 1.0], frameon=False, xticks=[],
        yticks=[])
    librosa.display.specshow(mel, cmap='gray_r')
    buf = io.BytesIO()
    plt.savefig(buf, bbox_inches=None, pad_inches=0,
        format="jpg")
    plt.clf()
    plt.close()

    buf.seek(0)
    img_array = np.asarray(bytearray(buf.read()))
    img_col = cv2.imdecode(img_array, cv2.IMREAD_UNCHANGED)
    img_gray = cv2.cvtColor(img_col, cv2.COLOR_BGR2GRAY)
    buf.flush()
    buf.close()

    num_slices = int(img_gray.shape[1]/128)
    img_gray = img_gray[:,:(num_slices*128)]
    split_spectrum = np.hsplit(img_gray, num_slices)

    return split_spectrum, num_slices
```

Below is the code snippet for splitting of the NumPy array into test and train. The test data consists of 500 audio tracks to be used for the recommendation model evaluation.

```
X_test = np.array(X[start:stop])
y_test = np.array(y[start:stop])
name_test = np.array(name[start:stop])

X_train_par1 = np.array(X[:start])
X_train_par2 = np.array(X[stop:])
y_train_par1 = np.array(y[:start])
y_train_par2 = np.array(y[stop:])
name_train_par1 = np.array(name[:start])
name_train_par2 = np.array(name[stop:])

X_train = np.concatenate([X_train_par1, X_train_par2])
y_train = np.concatenate([y_train_par1, y_train_par2])
name_train = np.concatenate([name_train_par1, name_train_par2])

if not os.path.exists("./train"):
    os.mkdir("./train")
np.save("./train/features.npy", X_train)
np.save("./train/classes.npy", y_train)
np.save("./train/names.npy", name_train)

if not os.path.exists("./test"):
    os.mkdir("./test")
np.save("./test/features.npy", X_test)
np.save("./test/classes.npy", y_test)
np.save("./test/names.npy", name_test)
```

6 Neural Network Classification Models

Three neural network classification models are used to classify the genre and for feature engineering. The three model Python codes and their corresponding architectures are listed below.

6.1 Convolutional Neural Network

Below is the code snippet for the Convolutional Neural Network model

```
cnn = Sequential(name="CNN")
cnn.add(Conv2D(filters=64, kernel_size=[7, 7],
    kernel_initializer=initializers.he_normal(
    seed=1), activation="relu"))  # Dim = (122x122x64)
cnn.add(BatchNormalization())
cnn.add(AveragePooling2D(pool_size=[2, 2], strides=2))
    # Dim = (61x61x64)
```

```
cnn.add(Conv2D(filters=128, kernel_size=[7, 7], strides=2,
    kernel_initializer=initializers.he_normal(seed=1),
    activation="relu"))  # Dim = (28x28x128)
cnn.add(BatchNormalization())
cnn.add(AveragePooling2D(pool_size=[2, 2], strides=2))
    # Dim = (14x14x128)

cnn.add(Conv2D(filters=256, kernel_size=[3, 3],
    kernel_initializer=initializers.he_normal(
    seed=1), activation="relu"))  # Dim = (12x12x256)
cnn.add(BatchNormalization())
cnn.add(AveragePooling2D(pool_size=[2, 2], strides=2))
    # Dim = (6x6x256)

cnn.add(Conv2D(filters=512, kernel_size=[3, 3],
    kernel_initializer=initializers.he_normal(
    seed=1), activation="relu"))  # Dim = (4x4x512)
cnn.add(BatchNormalization())
cnn.add(AveragePooling2D(pool_size=[2, 2], strides=2))
    # Dim = (2x2x512)

cnn.add(BatchNormalization())
cnn.add(Flatten())  # Dim = (2048)

cnn.add(BatchNormalization())
cnn.add(Dropout(0.6))

cnn.add(Dense(1024, activation="relu",
    kernel_initializer=initializers.he_normal(seed=1)))
    # Dim = (1024)
cnn.add(Dropout(0.5))

cnn.add(Dense(256, activation="relu",
    kernel_initializer=initializers.he_normal(seed=1)))
    # Dim = (256)
cnn.add(Dropout(0.25))

cnn.add(Dense(64, activation="relu",
    kernel_initializer=initializers.he_normal(seed=1)))
    # Dim = (64)

cnn.add(Dense(32, activation="relu",
    kernel_initializer=initializers.he_normal(seed=1)))
    # Dim = (32))

cnn.add(Dense(8, activation="softmax",
    kernel_initializer=initializers.he_normal(seed=1)))
```

Below is the Convolutional Neural Network architecture

```
Training CNN...
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (32, 122, 122, 64)        3200
_________________________________________________________________
batch_normalization (BatchNo (32, 122, 122, 64)        256
_________________________________________________________________
average_pooling2d (AveragePo (32, 61, 61, 64)          0
_________________________________________________________________
conv2d_1 (Conv2D)            (32, 28, 28, 128)         401536
_________________________________________________________________
batch_normalization_1 (Batch (32, 28, 28, 128)         512
_________________________________________________________________
average_pooling2d_1 (Average (32, 14, 14, 128)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (32, 12, 12, 256)         295168
_________________________________________________________________
batch_normalization_2 (Batch (32, 12, 12, 256)         1024
_________________________________________________________________
average_pooling2d_2 (Average (32, 6, 6, 256)           0
_________________________________________________________________
conv2d_3 (Conv2D)            (32, 4, 4, 512)           1180160
_________________________________________________________________
batch_normalization_3 (Batch (32, 4, 4, 512)           2048
_________________________________________________________________
average_pooling2d_3 (Average (32, 2, 2, 512)           0
_________________________________________________________________
batch_normalization_4 (Batch (32, 2, 2, 512)           2048
_________________________________________________________________
flatten (Flatten)            (32, 2048)                0
_________________________________________________________________
batch_normalization_5 (Batch (32, 2048)                8192
_________________________________________________________________
dropout (Dropout)            (32, 2048)                0
_________________________________________________________________
dense (Dense)                (32, 1024)                2098176
_________________________________________________________________
dropout_1 (Dropout)          (32, 1024)                0
_________________________________________________________________
dense_1 (Dense)              (32, 256)                 262400
_________________________________________________________________
dropout_2 (Dropout)          (32, 256)                 0
_________________________________________________________________
dense_2 (Dense)              (32, 64)                  16448
_________________________________________________________________
```

```
dense_3 (Dense)                 (32, 32)                      2080
_________________________________________________________________
dense_4 (Dense)                 (32, 8)                       264
=================================================================
Total params: 4,273,512
Trainable params: 4,266,472
Non-trainable params: 7,040
```

6.2 Convolutional Neural Network with Long Short-Term Memory

Below is the code snippet for the Convolutional Neural Network with Long Short-Term Memory model

```
cnn_lstm = Sequential(name="CNNLSTM")
cnn_lstm.add(Conv2D(filters=64, kernel_size=[7, 7],
    kernel_initializer=initializers.he_normal(
    seed=1), activation="relu"))  # Dim = (122x122x64)
cnn_lstm.add(BatchNormalization())
cnn_lstm.add(AveragePooling2D(pool_size=[2, 2], strides=2))
    # Dim = (61x61x64)

cnn_lstm.add(Conv2D(filters=128, kernel_size=[7, 7], strides=2,
    kernel_initializer=initializers.he_normal(seed=1),
    activation="relu"))  # Dim = (28x28x128)
cnn_lstm.add(BatchNormalization())
# Dim = (14x14x128)
cnn_lstm.add(AveragePooling2D(pool_size=[2, 2], strides=2))

cnn_lstm.add(Conv2D(filters=256, kernel_size=[3, 3],
    kernel_initializer=initializers.he_normal(seed=1),
    activation="relu"))  # Dim = (12x12x256)
cnn_lstm.add(BatchNormalization())
cnn_lstm.add(AveragePooling2D(pool_size=[2, 2], strides=2))
    # Dim = (6x6x256)

cnn_lstm.add(Conv2D(filters=512, kernel_size=[3, 3],
    kernel_initializer=initializers.he_normal(seed=1),
    activation="relu"))  # Dim = (4x4x512)
cnn_lstm.add(BatchNormalization())
cnn_lstm.add(AveragePooling2D(pool_size=[2, 2], strides=2))
    # Dim = (2x2x512)

cnn_lstm.add(BatchNormalization())
cnn_lstm.add(Dropout(0.6))

cnn_lstm.add(Reshape((512, -1)))
cnn_lstm.add(Permute((2, 1)))
```

```
cnn_lstm.add(LSTM(128, return_sequences=True,
    input_shape=(128, 128, 1)))
cnn_lstm.add(LSTM(128, input_shape=(128, 128, 1)))

cnn_lstm.add(Dense(1024, activation="relu",
    kernel_initializer=initializers.he_normal(seed=1)))
    # Dim = (1024)
cnn_lstm.add(Dropout(0.5))

cnn_lstm.add(Dense(256, activation="relu",
    kernel_initializer=initializers.he_normal(seed=1)))
    # Dim = (256)
cnn_lstm.add(Dropout(0.25))

cnn_lstm.add(Dense(64, activation="relu",
    kernel_initializer=initializers.he_normal(seed=1)))
    # Dim = (64)
cnn_lstm.add(Dense(32, activation="relu",
    kernel_initializer=initializers.he_normal(seed=1)))
    # Dim = (32)
cnn_lstm.add(Dense(8, activation="softmax",
    kernel_initializer=initializers.he_normal(seed=1)))
```

Below is the Convolutional Neural Network with Long Short-Term Memory architecture

```
Training CNNLSTM...
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (32, 122, 122, 64)        3200
_________________________________________________________________
batch_normalization_6 (Batch (32, 122, 122, 64)        256
_________________________________________________________________
average_pooling2d_4 (Average (32, 61, 61, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (32, 28, 28, 128)         401536
_________________________________________________________________
batch_normalization_7 (Batch (32, 28, 28, 128)         512
_________________________________________________________________
average_pooling2d_5 (Average (32, 14, 14, 128)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (32, 12, 12, 256)         295168
_________________________________________________________________
batch_normalization_8 (Batch (32, 12, 12, 256)         1024
_________________________________________________________________
average_pooling2d_6 (Average (32, 6, 6, 256)           0
_________________________________________________________________
conv2d_7 (Conv2D)            (32, 4, 4, 512)           1180160
```

```
batch_normalization_9 (Batch (32, 4, 4, 512)           2048
average_pooling2d_7 (Average (32, 2, 2, 512)           0
batch_normalization_10 (Batc (32, 2, 2, 512)           2048
dropout_3 (Dropout)          (32, 2, 2, 512)           0
reshape (Reshape)            (32, 512, 4)              0
permute (Permute)            (32, 4, 512)              0
lstm (LSTM)                  (32, 4, 128)              328192
lstm_1 (LSTM)                (32, 128)                 131584
dense_5 (Dense)              (32, 1024)                132096
dropout_4 (Dropout)          (32, 1024)                0
dense_6 (Dense)              (32, 256)                 262400
dropout_5 (Dropout)          (32, 256)                 0
dense_7 (Dense)              (32, 64)                  16448
dense_8 (Dense)              (32, 32)                  2080
dense_9 (Dense)              (32, 8)                   264
=================================================================
Total params: 2,759,016
Trainable params: 2,756,072
Non-trainable params: 2,944
```

6.3 Convolutional Neural Network with Bidirectional Long Short-Term Memory

Below is the code snippet for the Convolutional Neural Network with Bidirectional Long Short-Term Memory model

```
cnn_bi_lstm = Sequential(name="CNNBiLSTM")
cnn_bi_lstm.add(Conv2D(filters=64, kernel_size=[7, 7],
    kernel_initializer=initializers.he_normal(seed=1),
    activation="relu"))  # Dim = (122x122x64)
cnn_bi_lstm.add(BatchNormalization())
cnn_bi_lstm.add(AveragePooling2D(
    pool_size=[2, 2], strides=2))  # Dim = (61x61x64)
```

```
cnn_bi_lstm.add(Conv2D(filters=128, kernel_size=[7, 7],
    strides=2, kernel_initializer=initializers.he_normal(
    seed=1), activation="relu"))  # Dim = (28x28x128)
cnn_bi_lstm.add(BatchNormalization())
cnn_bi_lstm.add(AveragePooling2D(
    pool_size=[2, 2], strides=2))  # Dim = (14x14x128)

cnn_bi_lstm.add(Conv2D(filters=256, kernel_size=[3, 3],
    kernel_initializer=initializers.he_normal(seed=1),
    activation="relu"))  # Dim = (12x12x256)
cnn_bi_lstm.add(BatchNormalization())
cnn_bi_lstm.add(AveragePooling2D(
    pool_size=[2, 2], strides=2))  # Dim = (6x6x256)

cnn_bi_lstm.add(Conv2D(filters=512, kernel_size=[3, 3],
    kernel_initializer=initializers.he_normal(seed=1),
    activation="relu"))  # Dim = (4x4x512)
cnn_bi_lstm.add(BatchNormalization())
cnn_bi_lstm.add(AveragePooling2D(
    pool_size=[2, 2], strides=2))  # Dim = (2x2x512)

cnn_bi_lstm.add(BatchNormalization())
cnn_bi_lstm.add(Dropout(0.6))

cnn_bi_lstm.add(Reshape((512, -1)))
cnn_bi_lstm.add(Permute((2, 1)))
cnn_bi_lstm.add(Bidirectional(LSTM(128,
    return_sequences=True,input_shape=(128, 128, 1))))
cnn_bi_lstm.add(LSTM(128, input_shape=(128, 128, 1)))

cnn_bi_lstm.add(Dense(1024, activation="relu",
    kernel_initializer=initializers.he_normal(seed=1)))
    # Dim = (1024)

cnn_bi_lstm.add(Dropout(0.5))
cnn_bi_lstm.add(Dense(256, activation="relu",
    kernel_initializer=initializers.he_normal(seed=1)))
    # Dim = (256)
cnn_bi_lstm.add(Dropout(0.25))
cnn_bi_lstm.add(Dense(64, activation="relu",
    kernel_initializer=initializers.he_normal(seed=1)))
    # Dim = (64)
cnn_bi_lstm.add(Dense(32, activation="relu",
    kernel_initializer=initializers.he_normal(seed=1)))
    # Dim = (32)
cnn_bi_lstm.add(Dense(8, activation="softmax",
    kernel_initializer=initializers.he_normal(seed=1)))
```

Below is the Convolutional Neural Network with Bidirectional Long Short-Term Memory architecture

```
Training CNNBiLSTM...
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (32, 122, 122, 64)        3200
_________________________________________________________________
batch_normalization_11 (Batc (32, 122, 122, 64)        256
_________________________________________________________________
average_pooling2d_8 (Average (32, 61, 61, 64)          0
_________________________________________________________________
conv2d_9 (Conv2D)            (32, 28, 28, 128)         401536
_________________________________________________________________
batch_normalization_12 (Batc (32, 28, 28, 128)         512
_________________________________________________________________
average_pooling2d_9 (Average (32, 14, 14, 128)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (32, 12, 12, 256)         295168
_________________________________________________________________
batch_normalization_13 (Batc (32, 12, 12, 256)         1024
_________________________________________________________________
average_pooling2d_10 (Averag (32, 6, 6, 256)           0
_________________________________________________________________
conv2d_11 (Conv2D)           (32, 4, 4, 512)           1180160
_________________________________________________________________
batch_normalization_14 (Batc (32, 4, 4, 512)           2048
_________________________________________________________________
average_pooling2d_11 (Averag (32, 2, 2, 512)           0
_________________________________________________________________
batch_normalization_15 (Batc (32, 2, 2, 512)           2048
_________________________________________________________________
dropout_6 (Dropout)          (32, 2, 2, 512)           0
_________________________________________________________________
reshape_1 (Reshape)          (32, 512, 4)              0
_________________________________________________________________
permute_1 (Permute)          (32, 4, 512)              0
_________________________________________________________________
bidirectional (Bidirectional (32, 4, 256)              656384
_________________________________________________________________
lstm_3 (LSTM)                (32, 128)                 197120
_________________________________________________________________
dense_10 (Dense)             (32, 1024)                132096
_________________________________________________________________
dropout_7 (Dropout)          (32, 1024)                0
_________________________________________________________________
dense_11 (Dense)             (32, 256)                 262400
```

```
_________________________________________________________________
dropout_8 (Dropout)          (32, 256)                 0
_________________________________________________________________
dense_12 (Dense)             (32, 64)                  16448
_________________________________________________________________
dense_13 (Dense)             (32, 32)                  2080
_________________________________________________________________
dense_14 (Dense)             (32, 8)                   264
=================================================================
Total params: 3,152,744
Trainable params: 3,149,800
Non-trainable params: 2,944
```

7 Recommendation Model

The content-based recommendation system is used with Cosine similarity. Following is the Python code for building the model, the model's genre accuracy, and the recommendations for the anchor song.

7.1 Content-Based Recommendation Mode Code Snippet

Below is the code snippet for the recommendation model using Cosine Similarity

```
for model_name, model_path in best_models.items():

    print(f"Currently evaluating the recommender engine using
    {model_name} model")

    #load model for recommendation
    loaded_model = load_model(model_path,
    custom_objects=dependencies)
    loaded_model.set_weights(loaded_model.get_weights())

    #use the bottleneck layers as it contains all the latent
    features of each song
    matrix_size = loaded_model.layers[-2].output.shape[1]
    new_model = Model(loaded_model.inputs,
    loaded_model.layers[-2].output)

    #Get the average accuracy of genre recommendations per
    genre
    test_songs = defaultdict(init_dict)

    #get latent features from the bottleneck layer for
    each song
    print("Calculating latent features for each song...")
    total = len(images)
```

```
for image, name in tqdm(zip(images, names),
total=total, file=sys.stdout):
    img_reshaped = np.expand_dims(image, axis=0)
    pred = new_model.predict(img_reshaped)
    test_songs[name] += pred

test_songs_scaled = {k:v/10 for k,v in test_songs.items()}

top_k = [5, 10] #no of rec
for k in top_k:
    genre_based_accuracy = defaultdict(int)

    #iterate through each song in the test dataset and the
    cosine distance from all songs and get k nearest songs
    print("Obtaining recommendation accuracy for each
    genre...")
    for song, features in tqdm(test_songs_scaled.items(),
    file=sys.stdout):
        distance_array = list(map(lambda x:
        cosine_similarity(features, x),
        test_songs_scaled.values()))
        distance_array_with_songs = list(zip(distance_array,
        test_songs_scaled.keys()))
        #arranged in descending order
        final_array = sorted(distance_array_with_songs,
        key=lambda x: x[0], reverse=True)

        #get the genre of the k nearest songs and compare
        with the genre of the anchor song
        genre_list = list()
        for i in final_array[0:k+1]:
          genre_index = np.where(names==i[1])[0]
          genre_list.append(genres[genre_index][0])

        anchor_genre = [genre_list[0]]*k
        recom_genre = genre_list[1:]

        accuracy = sum([x==y for x,y in
        zip(anchor_genre,recom_genre)])/k

        #Add the accuracy to the list
        genre_based_accuracy[genre_list[0]] += accuracy

    #scale the list by count of songs in each genre
    genre_based_accuracy_percentage =
    {k:round(((v/int(all_genre_counts.get(k)/10)))*100,2)
    for k, v in genre_based_accuracy.items()}
```

```
        print(f"For {model_name} the genre based accuracy for
        top {k} recommendations are")
        print(genre_based_accuracy_percentage)
```

7.2 Recommendation Model Evaluation

Below is the recommendation model evaluation based on genre accuracy

```
Currently evaluating the recommender engine using CNN model
Calculating latent features for each song...
100%|| 5000/5000 [02:38<00:00, 31.60it/s]
Obtaining recommendation accuracy for each genre...
100%|| 500/500 [00:05<00:00, 97.20it/s]
For CNN the genre based accuracy for top 5
recommendations are
{'International': 97.52, 'Experimental': 79.07, 'Pop': 77.35,
'Electronic': 90.35, 'Hip-Hop': 92.86, 'Folk': 80.0,
'Rock': 89.58, 'Instrumental': 23.33}
Obtaining recommendation accuracy for each genre...
100%|| 500/500 [00:05<00:00, 94.11it/s]
For CNN the genre based accuracy for top 10
recommendations are
{'International': 96.76, 'Experimental': 78.6, 'Pop': 77.95,
'Electronic': 91.18, 'Hip-Hop': 92.68, 'Folk': 76.77, 'Rock':
88.12, 'Instrumental': 18.33}
Currently evaluating the recommender engine using CNNLSTM model
Calculating latent features for each song...
100%|| 5000/5000 [02:38<00:00, 31.50it/s]
Obtaining recommendation accuracy for each genre...
100%|| 500/500 [00:05<00:00, 96.18it/s]
For CNNLSTM the genre based accuracy for top 5
recommendations are
{'International': 99.05, 'Experimental': 94.19, 'Pop': 92.29,
'Electronic': 96.24, 'Hip-Hop': 97.5, 'Folk': 94.19,
'Rock': 96.25, 'Instrumental': 83.33}
Obtaining recommendation accuracy for each genre...
100%|| 500/500 [00:05<00:00, 93.72it/s]
For CNNLSTM the genre based accuracy for top 10
recommendations are
{'International': 98.76, 'Experimental': 93.6, 'Pop': 90.72,
'Electronic': 97.06, 'Hip-Hop': 97.32, 'Folk': 92.26,
'Rock': 96.67, 'Instrumental': 48.33}
Currently evaluating the recommender engine using CNNBiLSTM
model
Calculating latent features for each song...
100%|| 5000/5000 [02:42<00:00, 30.83it/s]
Obtaining recommendation accuracy for each genre...
100%|| 500/500 [00:05<00:00, 96.03it/s]
For CNNBiLSTM the genre based accuracy for top 5
```

```
recommendations are
{'International': 99.24, 'Experimental': 82.33, 'Pop': 91.08,
'Electronic': 94.35, 'Hip-Hop': 94.29, 'Folk': 96.77,
'Rock': 95.42, 'Instrumental': 46.67}
Obtaining recommendation accuracy for each genre...
100%|| 500/500 [00:05<00:00, 94.26it/s]
For CNNBiLSTM the genre based accuracy for top 10
recommendations are
{'International': 99.43, 'Experimental': 80.93, 'Pop': 90.24,
'Electronic': 95.29, 'Hip-Hop': 94.46, 'Folk': 94.52,
'Rock': 94.58, 'Instrumental': 35.0}
```

7.3 Recommendations Based on Anchor Track Code Snippet

Below is the code snippet to recommend songs based on anchor track

```
print(*np.unique(names),sep="\n")
song = input("Kindly select a song from above: ")
#load model for recommendation
loaded_model = load_model(best_models["CNNLSTM"],
    custom_objects=dependencies)
loaded_model.set_weights(loaded_model.get_weights())

#use the bottleneck layers as it contains all the latent
features of each song
matrix_size = loaded_model.layers[-2].output.shape[1]
new_model = Model(loaded_model.inputs,
    loaded_model.layers[-2].output)

#Get the average accuracy of genre recommendations per genre
test_songs = defaultdict(init_dict)

print("Calculating latent features for each song...")
total = len(images)
for image, name in tqdm(zip(images, names), total=total,
    file=sys.stdout):
    img_reshaped = np.expand_dims(image, axis=0)
    pred = new_model.predict(img_reshaped)
    test_songs[name] += pred

test_songs_scaled = {k:v/10 for k,v in test_songs.items()}

song_distance_array = list(map(lambda x:
    cosine_similarity(test_songs_scaled[song], x),
    test_songs_scaled.values()))
song_distance_array_with_songs = list(zip(song_distance_array,
    test_songs_scaled.keys()))
#arrange in descending order
probable_songs = sorted(song_distance_array_with_songs,
```

```
    key=lambda x: x[0], reverse=True)

rec_no=int(input("How many recommendations would you like: "))
for song in probable_songs[1:rec_no+1]:
  print(song[1])
```

7.4 5 Recommendations Based on Anchor Track

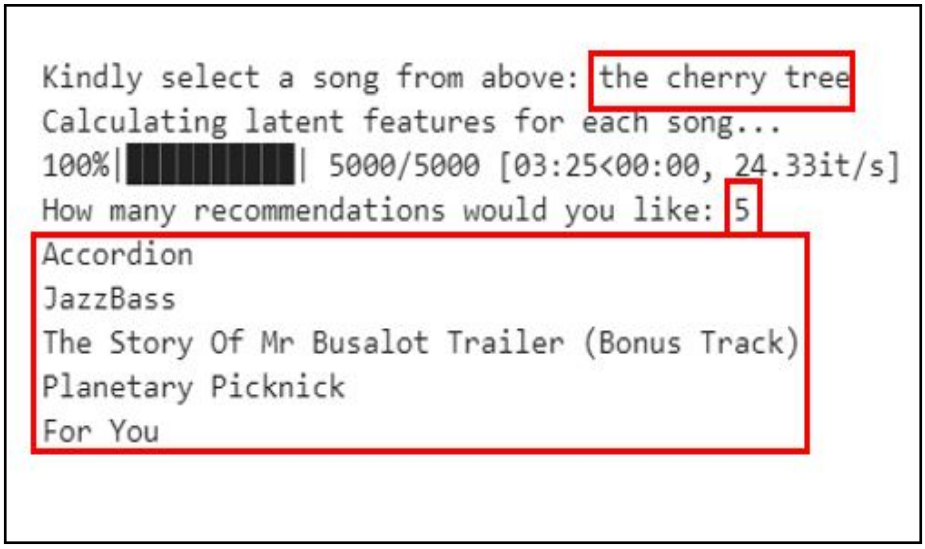

Figure 5: 5 Recommend Songs Based on Anchor Track

7.5 10 Recommendations Based on Anchor Track

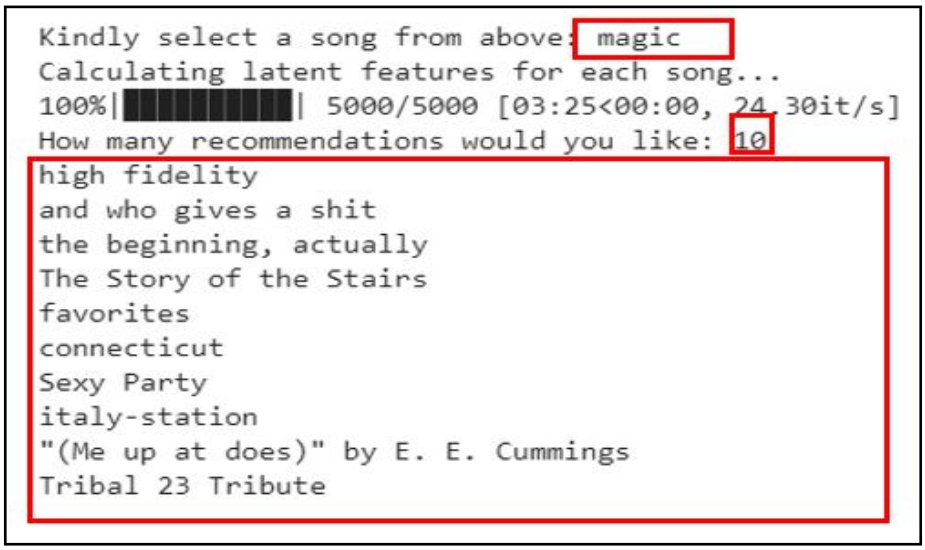

Figure 6: 10 Recommend Songs Based on Anchor Track

8 Code Implementation

The steps as seen in Figure 7 needs to be carried out to implement the code. The workspace folder is present in the ICT document and these steps are also mentioned in a readme.txt file attached with the workspace.

The following steps need to be carried out to implement the code for the music recommendation system

1. Download/Copy the MSc_Research_Project_Code.zip onto a local path and unzip it.

2. Download the fma_metadata.zip and fma_small.zip files from https://github.com/mdeff/fma

3. Create a folder called Dataset in MSc_Research_Project_Code.

4. Unzip fma_metadata.zip and fma_small.zip files into the Dataset folder in MSc_Research_Project_Code.

5. Check the requirements.txt file for the packages and libraries version requirements.

6. Run the eda.ipynb and preprocessing.ipynb files in Jupyter Notebook.

7. The preprocessing.ipynb file will create two new folders of test and train in MSc_Research_Project_Code.

8. 3 npy files will be generated in each of the folders. The 3 npy files are classes.npy, features.npy and names.npy.

9. Create a google account and upload the updated MSc_Research_Project_Code folder onto Google Drive. Get a Google Colab Pro account for running the code.

10. Run the train.ipynb file on Google Colab Pro. A checkpoint folder will be created by the system with subfolders for the 3 models.

11. Each of these folders will contain class weight files in h5 format corresponding to the epochs number. 29 h5 files will be generated.

12. You will be prompted to input the epoch at which the model performed the best(High accuracy and low loss which can be viewed in the console). The evaluation metrics will be printed for it on the console.

13. You will be prompted to input 'Y' or 'N' if you want that model to be saved and used further. On inputting Y the model will be saved in a system created best_models folder.

If you input 'N' then you will be prompted to input another epoch and then select 'Y' or 'N' for saving it.

14. Steps 10 and 11 will be carried out 3 times where the user will be prompted for the 3 models.

15. Run the recommender.ipynb file in Google Colab Pro. The user will be prompted to input an anchor song from the above-mentioned songs in the console and also input the number of songs to be recommended. The recommended songs will be printed on the console.

Figure 7: Detailed Steps for Code Implementation

References

Defferrard, M., Benzi, K., Vandergheynst, P. and Bresson, X. (2016). Fma: A dataset for music analysis.