B.Tech. Project Report
CO-414
on

DEEP CONTENT BASED MUSIC RECOMMENDATION SYSTEM

BY

AHMAD ALI ABDILAH (2140012)
KAPIL RAMNANI (2130011)
UJWAL SASIKUMAR (2130008)

Under the Supervision of
Dr. Minakshi Sharma, Assistant Professor

DEPARTMENT OF COMPUTER ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
KURUKSHETRA – 136119, HARYANA (INDIA)
Dec 2017 – April 2018

CERTIFICATE

I hereby certify that the work which is being presented in this B.Tech. Major Project (CO-414) report entitled **"DEEP CONTENT BASED MUSIC RECOMMENDATION SYSTEM",** in partial fulfilment of the requirements for the award of the **Bachelor of Technology in Computer Engineering** is an authentic record of my own work carried out during a period from December 2017 to April 2018 under the supervision of **Dr. Minakshi Sharma**, Assistant Professor, Computer Engineering Department.

The matter presented in this project report has not been submitted for the award of any other degree elsewhere.

Signature of Candidate

AHMAD ALI ABDILAH (2140012)

KAPIL RAMNANI (2130011)

UJWAL SASIKUMAR (2130008)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date:

Signature of Supervisor

Dr. Minakshi Sharma

Assistant Professor

TABLE OF CONTENTS

Section No.	TITLE	Page no.
	ABSTRACT	1
I	INTRODUCTION	2
II	LITERATURE SURVEY	5
	II.I Content-Based Recommendation System	5
	II.II Convolutional Neural Network (CNN)	8
III	CONTRIBUTION	9
	III.I Content Based Music Recommendation System	9
	III.II Song Labeling Convolutional Neural Network	11
IV	DATA FLOW DIAGRAM	12
	IV.I Level 0 DFD	12
	IV.II Level 1 DFD	12
	IV.III Level 2 DFD	13
V	RESULTS/ OBSERVATION	15
	CONCLUSION AND FUTURE PLAN	17
	REFERENCES	18
APPENDIX:		
A	COMPLETE CONTRIBUTARY SOURCE CODE	19

ABSTRACT

In this new era where production of music is completely digitalized in the market as well as taking into account demand, automatic music recommendation system positions itself as a highly relevant problem. while the majority of recommender systems are built on collaborative filtering the disadvantage of such approach is that it fails due the cold start problem as with the introduction of new songs in the system and the unavailability of no usage data it is not ideal and inefficient for new as well as songs with less popularity. To get around this major disadvantage we propose to overcome this with the advent of latent factor model for recommendation, and produce predictively the latent factors from the music audio to compensate the unavailability of usage data. We use the spectrogram representation of the audio signals with deep convolutional neural network on combined dataset from various sources. Hence it is apparent with the results concluding sensible recommendations by introducing predicted latent factors, despite huge semantic void involving the characteristics of music that acts a hindrance between the user preferences/choices and the associated music signal.

I. INTRODUCTION

It is apparent from the last few years that the music market is having a change in its distribution methodology's by focusing and targeting digital distribution via online music stores and other popular music applications such as iTunes, Groove shark and Spotify and Google play. As a result, automatic music recommendation system positions itself as a highly relevant problem which helped users to explore as well as attain music that is in liking and parallel with their preferences, also favors online stores to direct their sales towards the right direction. Even after the studies and exploiting the music recommendations system the problem narrows down to diverse factors such as style, genres, social, geographical factors that has a solid connection with user preferences. while the recommendation especially for an individual song involves extensively different items this count can be decremented by just producing albums and artist recommendations however this approach fails to be incompatible with the systems with certain intentions such as automatic music playlist players and having no regards to the fact that repertoire of artist is very unlikely and uncommonly homogenous: users may prefer and like certain songs better than the rest users .The user pattern analysis is one of the conventional method where the collective information about the songs used or the ratings put forth the user interests and likings as well as how the items are in parallel in relation with each other and this approach is known as collaborative filtering approach. Additionally, there are other approaches where we can obtain the preferences of the users with the metadata and item content. The ideal opinion shared by many people is that collaborative filtering will dominate content based recommendation provided the usage data is at hand [1]. Collaborative filtering has a weakness of cold start problem: where new and unpopular items are given no regards as items with niche interest face the problem on recommendation due to data scarcity.

The inspiration of performing this project is based on a paper that states "… predicting latent factors from music audio is a viable method for recommending new and unpopular music. [5]" In that paper, the researchers are trying to give recommendation based on the latent factors obtained from the raw file of the song. Further, upon researching more regarding music latent factors, we found a paper that explains song auto tagging by using latent factors of the song [6].

MOTIVATION

The development of technology has brought up some solutions to a lot of problems. While the industries of music are growing, a demand comes to create a better experience for the users in consuming the music.

An analysis on preferences between users is one of the conventional methods where the collective information about the songs listened or the ratings put forth the user interests and likings as well as how the items are in parallel in relation with each other and this approach is known as collaborative filtering approach. But this technique suffers from cold start problem and the void created without the context of what song features make a user like them seems like a problem that we would like to address and attempt to propose a solution: content-based music recommendation system.

When selecting a content-based music recommendation, another problem arises: defining song features that are used as parameter in the recommendation. To tackle this problem, we associate labels to each song which defines the song implicitly. But the labeling of songs would be high in cost and time consuming, not to mention the hardship to have standardized labels. Upon doing research, deep learning seems to be an efficient solution to solve this problem. Deep learning also seems like a natural choice of study which we infer will be much interest and challenging.

As a team with keen interest in machine learning and the future scope of recommender systems our team has been dedicated in this field since our work began. Through studies and exploitation of research papers our team was led to ponder many questions on the possibilities of introducing novel technical approaches to overcome the current recommender system drawbacks. This project has given us the opportunity to exploit and bridge our knowledge in the domain of machine learning and recommendation systems.

Our knowledge at the initial stages of project development was casual about the recommender system, deep learning and data mining techniques such as cleaning and transformation of data. Without doubt this is the area of interest which we wish to pursue in the future as far as our career is concerned.

The consensus of our opinion is with the advent of deep learning and with a proposed example of possible extension of the development of the system by integrating mood detection. We have proposed a music recommendation that will fill that gap between the user mood and the relative music he/she would prefer to hear that was inferred, which was decided after from many reading many research papers and reports. On the theoretical aspect we have exploited much into the novelty for recommendation systems and other advanced techniques to build an efficient music recommendation system.

II. LITERATURE SURVEY

III.I. Content-Based Recommendation System

Systems implementing a content-based recommendation approach analyze a set of documents and/or descriptions of items previously rated by a user, and build a model or profile of user interests based on the features of the objects rated by that user. The profile is a structured representation of user interests, adopted to recommend new interesting items. The recommendation process basically consists in matching up the attributes of the user profile against the attributes of a content object. The result is a relevance judgment that represents the user's level of interest in that object. If a profile accurately reflects user references, it is of tremendous advantage for the effectiveness of an information access process [2].

Content Based Recommendation is system is a type of recommendation system that gives items that may be preferred by the user of the system by considering the features of the items. Different from collaborative Filtering technique in recommendation system in which giving recommendation based on the common items rated by related users, therefore have a critical issue when the item is never listened by anyone in the system yet which results unreachable by the system, content based recommendation does not suffer from this problem [3]. This advantage of this technique exist because of the feature of an item is provided and similarity of each items in the system, therefore, can be deduced.

Another strength of content based recommendation system is it can give recommendation based on the context. Taking our project as example, giving recommendation to a person can be personalized based on his rating history on songs and the current mood of the user. The ability to adapt recommendation based on the context makes it is possible for the system to give a better recommendation to the users.

A content based recommendation system consist of 3 major steps: constructing weighing factors of feature of items, constructing user's profile, and computing similarity. The similarity computation is designed to produce larger values for similarities between user's profile and weighing factors of the items. Then the list of computed similarity is returned in descending order as a recommendation.

III.I.I. Construction of Weighing Factor of Items

Each item has feature that defines the characteristic of the items. This characteristic is taken into account in recommending items to the users. There are many ways in representing feature of items, one of them is by a list of labels associated with an item. To compute weighing factor of an item when labels are associated, we can one of techniques called TF-IDF.

TF-IDF is a numerical statistic that is intended to reflect how important a label is to an item in a collection or items. In our project the items are songs. In calculating TF-IDF, 2 steps are to be performed: calculating TF (term Frequency) of the labels and calculating IDF (Inverse Document Frequency) related ot labels. The TF and IDF values acquired from each steps then multiplied to get TF-IDF value.

Term Frequency

Term frequency is basically means how many times a label appear for a song. The appearance of a label in a song indicates how many relevant a label to a song. Suppose a song is associated with a 'slow' label that indicates a song's tempo is slow. Multiple appearance of label 'slow' increases the TF of this label with respect to the song.

Inverse Document Frequency

Inverse Document Frequency is a a way to normalize the TF value of a label in a song. A label that appears in multiple song is less relevant (have less IDF score) than the one that appears only in a few song. This phase is essential to weigh down the terms that apears in too much songs. Suppose a label 'slow' appears in 5 songs, while 'pop' label (indicates the genre of the song) only appear in one of the song. In this case, the label 'song' will have less weigh than 'pop', and the system will concentrate more on higher IDF values.

III.I.II. Construction of user's profile

User profile in a recommendation system is important to produce a personalized recommendation for each user. It gives a better experience to the users who are using our system.

One of the common ways to construct a user's profile is using explicit technique, which is based on the rating a user gives to an item, in this case song. Ratings the users give for items are collected, and based on those ratings, user's profile is built.

The rating is then filtered according to a treshold, for example only to consider a positive rating. Then the features of the items rated positively by the user is then extracted. A weight of each labels/feature is then determined by how many times it appears in the items the user has rated positively. Accordingly then profile of all users in the system is constructed.

III.I.III. Computation of similarity

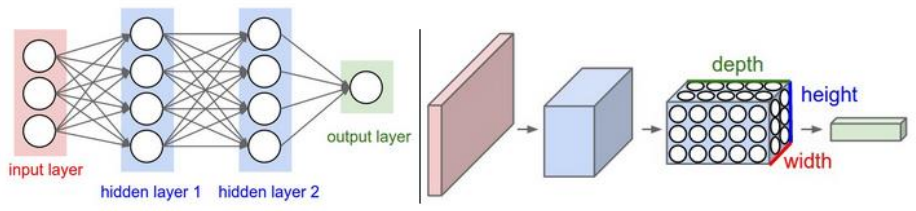
Similarity computation calculates how much relevant the user's profile with the item's features in the system. This phase is a final phase to serve a recommendation to the user. The item which has higher relevancy will have higher score and, thus places higher in the recommendation list. There are many techniques in computing a similarity of items, one of them is Cosine Similarity.

A measure of the similarity between two objects a and b, often used in information retrieval, consists in representing these objects in the form of two vectors xa and xb and computing the Cosine Vector (CV) (or Vector Space) similarity. The formula of cosine similarity is as follows:

$$\cos(\mathbf{x}_a, \mathbf{x}_b) = \frac{\mathbf{x}_a^\top \mathbf{x}_b}{||\mathbf{x}_a||\,||\mathbf{x}_b||}.$$

III.II. Convolutional Neural Network (CNN)

Convolutional Neural Network is a type of neural network that is usually used to enable a machine from a complex dataset, like image or audio file. The choice of using convolutional neural network on complex data is due to it's not fully connected structure of the hidden layers. It makes the structure of the neural network simpler and more efficient operations can be performed.



Convolutional networks were inspired by biological processes [4] in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

III. CONTRIBUTION

There are 2 major concepts we use to serve as a solution of the problem we proposed earlier as a project: Content-Based Recommendation System and Convolutional Neural Network (CNN). This both concepts together make a "deep content based music recommendation" possible.

IV.I. Content Based Recommendation System

<u>Pseudocode of the content based recommendation system:</u>

The Content-Based Recommendation is used to give a list of song recommendations to the user based on the user profile, song features and additionally special label that would be added as a query, like the mood of the song for example.

<u>Construction of Weighing Factor of Songs:</u>

```
1. song_data = Load song data and labels associated
//count label appearance
2. for each song_data
3.          TF[song][label]+=1
4. const = log10(unique labels)
5. for each song_data
6.          IDF[label]+=1
7. for each data in TF
8.          TF_IDF[song][label] = TF[song][label]*IDF[label]
9. return TF_IDF
```

Construction of user profile:

```
1. rating_data = load rating_data
2. user = extract user data from rating_data
3. user_distinct = drop multiple entry of user
4. for each user in user_distinct
5.        user_profile[user][label]+= TF_IDF[song][label]
6. return user_profile
```

Computation of Similarity:

```
1. rating_data = load rating_data
2. user = extract user data from rating_data
3. user_distinct = drop multiple entry of user
4. song_data = load song data
4. for each user in user_distinct
5.        distinct_song = extract distinct song related to the user from song data
6.        for each song in distinct_song
7.                sum_user_profile+=sum(user_profile)
8.                sum_label_weight+=sum(TF_IDF)
9.                sum_nominator+=user_profile[user]*TF_IDF[song]
10.similarity[user][song]=sum_nominator/(sum_user_profile*sum_label_weight)
11. return sort_descending(similarity)
```

IV.II. Song Labeling Convolutional Neural Network

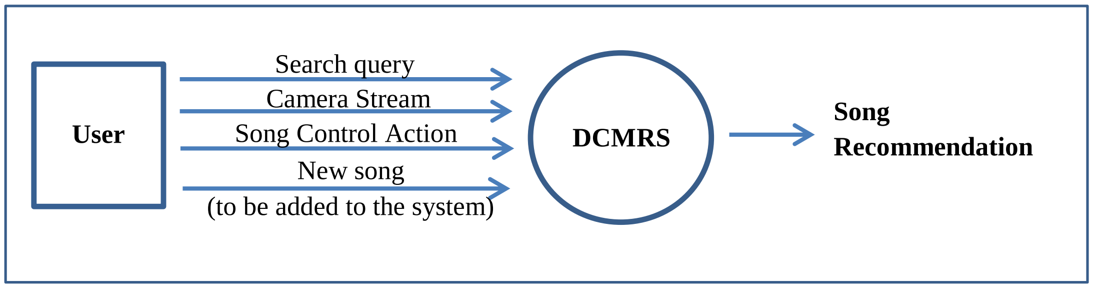
The convolutional neural network (CNN) is used as a way to associate labels to a new song automatically. It would enable a new song to be added to the system without specifying features of the song explicitly. This turns out would eliminate manual labor and will result a more standardized label for songs. This functionality would consist of construction of the CNN and training of the model.

<u>Pseudo code of Song Labeling Convolution Neural Network Model:</u>

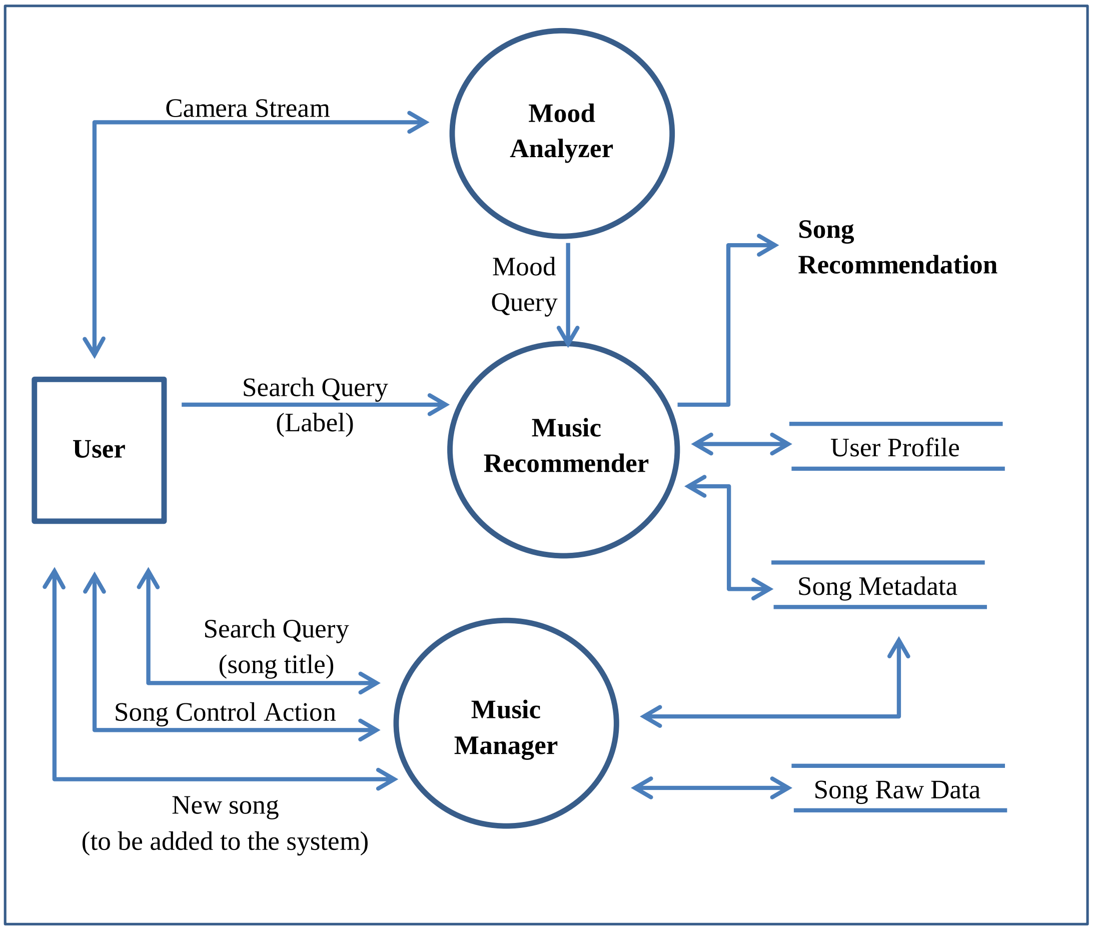
```
1. raw_songs = load raw songs
2. for each song in raw_songs
3.         song_features[song] = extract_feature(song)
4. input_layer = construct input layer with dimension (1, 96, 1366)
5. for i to 5
6.         hidden_layer[i] = Construct 2D convolutional layer
7.         hidden_layer[i] += BatchNormalization
8.         hidden_layer[i] += ELU
9.         hidden_layer[i] += MaxPooling2D
10. output_layer = construct fully connected output layer with dimension 50
11. CNN = merge(input_layer, hidden_layer, output_layer)
12. song_labels = load song labels data
13. //separating dataset into train and test set
14. epoch = 10
15. batch = 10
16. CNN.fit(song_features, song_labels, epoch, batch)
```

IV. DATA FLOW DIAGRAM

V.I. Level 0 DFD

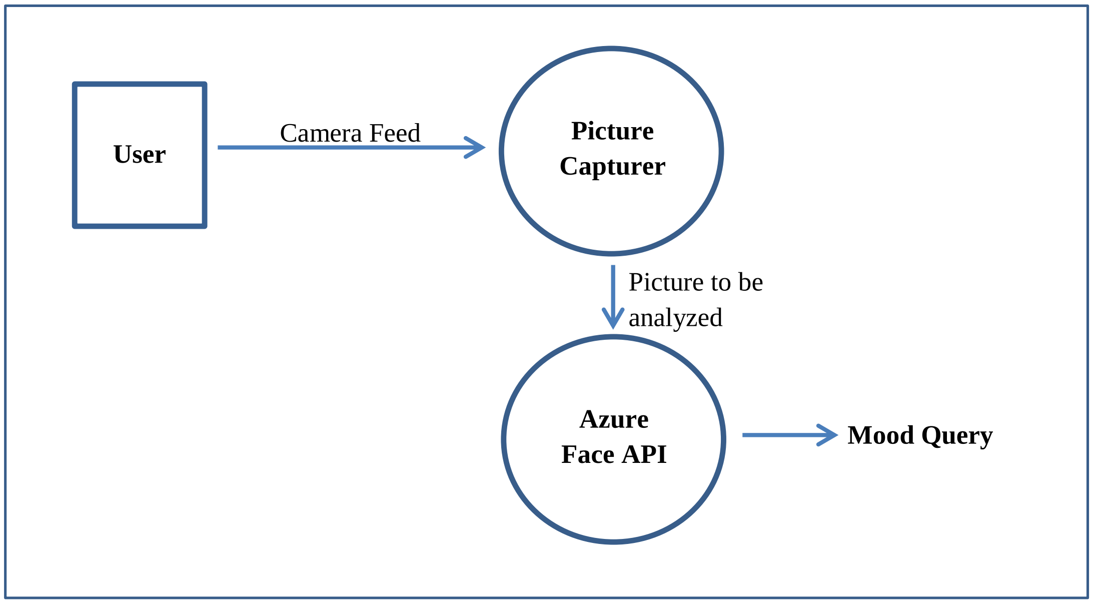


V.II. Level 1 DFD: Deep Content-Based Music Recommender System (DCMRS)

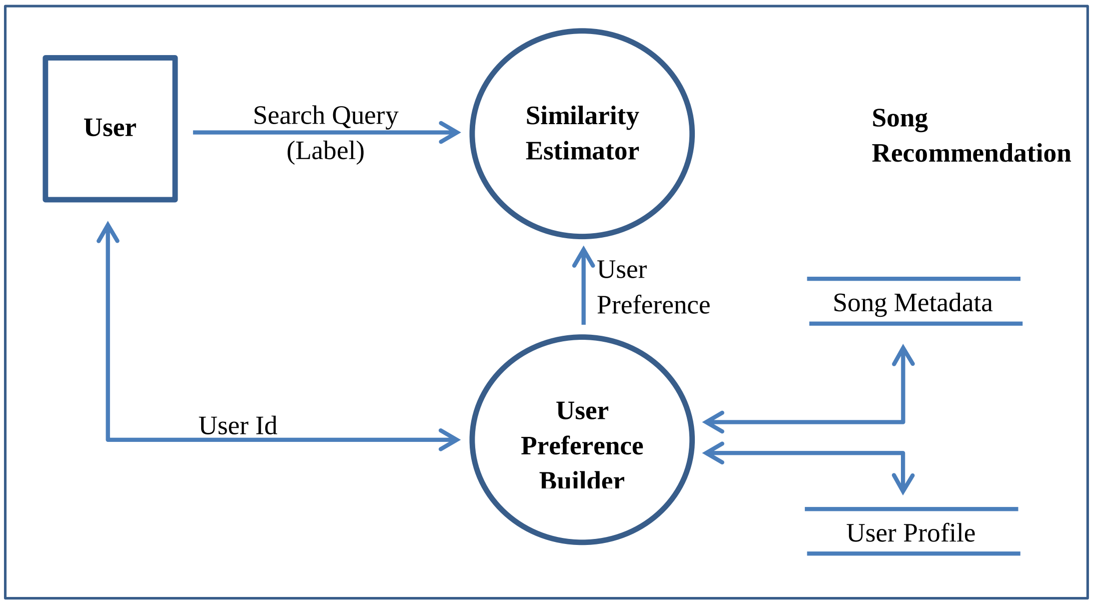


V.III. Level 2 DFD

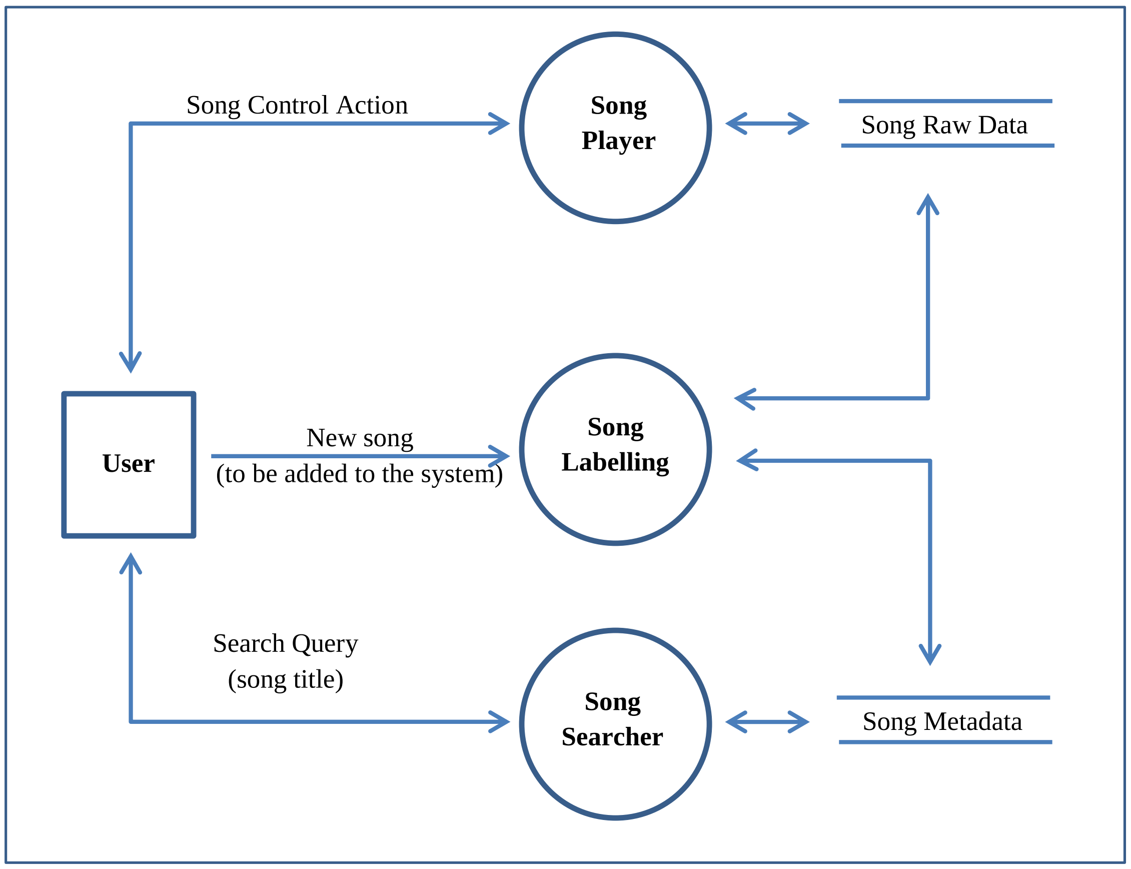
<u>Mood Analyzer</u>



<u>Music Recommender</u>

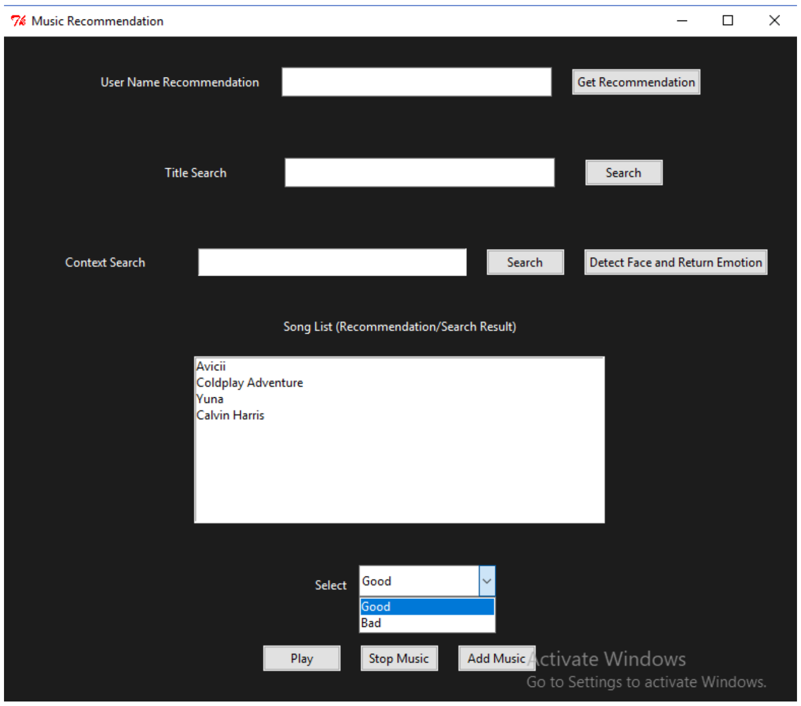


Music Manager



V. RESULTS/ OBSERVATIONS

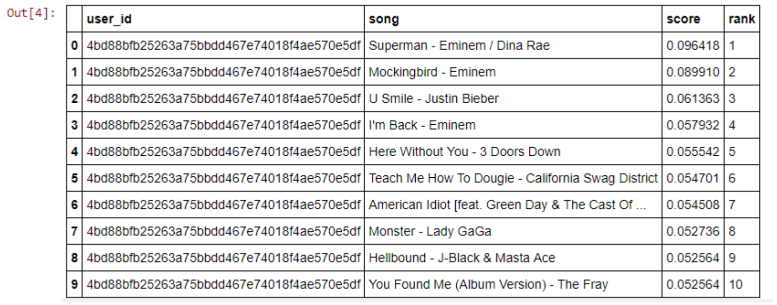
In the project we did not only implemented content based recommendation system, but also some of the collaborative filtering technique for the sake of comparison and showing practical difference between them. Besides that, we also created GUI to simulate the real world situation where a song recommendation system is used in the music player.



Avicii
Coldplay Adventure
Yuna
Calvin Harris

Good
Good
Bad

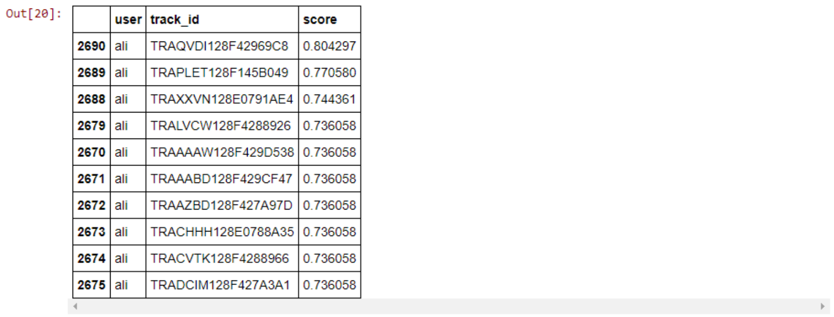
The Following is the result of the collaborative filtering technique in the recommendation system using Coocurance Matrix technique. This technique is based on the listen count of the song to recommend list of songs to the user. As other collaborative filtering techniques, this technique suffers from 'cold start' problem. The edge case would be: when a song has not been listened by anyone, the song will never come to recommendation, thus restricting the recommendation system to reach them.



Out[4]:

	user_id	song	score	rank
0	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Superman - Eminem / Dina Rae	0.096418	1
1	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Mockingbird - Eminem	0.089910	2
2	4bd88bfb25263a75bbdd467e74018f4ae570e5df	U Smile - Justin Bieber	0.061363	3
3	4bd88bfb25263a75bbdd467e74018f4ae570e5df	I'm Back - Eminem	0.057932	4
4	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Here Without You - 3 Doors Down	0.055542	5
5	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Teach Me How To Dougie - California Swag District	0.054701	6
6	4bd88bfb25263a75bbdd467e74018f4ae570e5df	American Idiot [feat. Green Day & The Cast Of ...	0.054508	7
7	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Monster - Lady GaGa	0.052736	8
8	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Hellbound - J-Black & Masta Ace	0.052564	9
9	4bd88bfb25263a75bbdd467e74018f4ae570e5df	You Found Me (Album Version) - The Fray	0.052564	10

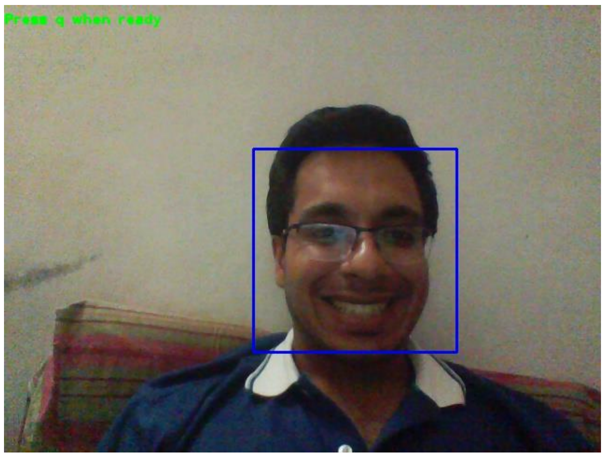
The next section we show the list of recommendation of songs in which a newly added song which never listened before come to recommendation list for the user, regardless that the song never listened by anyone before. By using this technique, we can increase the coverage of the recommendation to not only popular song, but also unlistened song. When implemented in real media industry, it will be beneficial to raise awareness of the user of songs he even did not know but still within the interest of the user (which is captured by user profile constructed).



Out[20]:

	user	track_id	score
2690	ali	TRAQVDI128F42969C8	0.804297
2689	ali	TRAPLET128F145B049	0.770580
2688	ali	TRAXXVN128E0791AE4	0.744361
2679	ali	TRALVCW128F4288926	0.736058
2670	ali	TRAAAAW128F429D538	0.736058
2671	ali	TRAAABD128F429CF47	0.736058
2672	ali	TRAAZBD128F427A97D	0.736058
2673	ali	TRACHHH128E0788A35	0.736058
2674	ali	TRACVTK128F4288966	0.736058
2675	ali	TRADCIM128F427A3A1	0.736058

To demonstrate the huge possibility of content based music recommendation, we also integrate a functionality to enable the system to recommend the user based on his mood captured from his face. This functionality is shown below:



VI. CONCLUSION AND FUTURE PLAN

The technique which is used typically in the song recommendation is using collaborative filtering. However, with the limitations of collaborative filtering in ‘cold start’ state and its inability to adapt with context, motivated us to perform this project. Here are the points we have achieved:

1. The solution we try to offer in this project is to help solving problem of cold start with collaborative filtering.
2. Our system can help to remove manual labor in labeling the song by predicting the labels suitable for the song.
3. Being context aware, our project extends the functionality of content-based recommendation system by integrating with a system that enables us to detect the mood of the person form the picture taken from his camera.

There are still a lot of possible enhancement in the project:

1. Running the neural network model on more dataset to increase its ability to predict labels and to increase its accuracy.
2. Popularity prediction of a song by analyzing the popularity song songs that closely similar to a new song.
3. Combining the content based recommender we have with collaborative filtering technique to achieve hybrid approach to get more varied recommendations.
4. Adding more functionality to input a context as a query to the system, for example voice based.

REFERENCES

[1]. Adomavicius,G., Tuzhilin,A. :Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering17(6), 734–749 (2005)

[2]. Ricci, F. (2011). Recommender systems handbook. New York: Springer.

[3]. Balabanovi´c, M., Shoham, Y.: Fab: Content-based, collaborative recommendation. Communications of the ACM40(3), 66–72 (1997)

[4]. Matusugu, Masakazu; Katsuhiko Mori; Yusuke Mitari; Yuji Kaneda (2003). "Subject independent facial expression recognition with robust face detection using a convolutional neural network" (PDF). Neural Networks. 16 (5): 555–559. doi:10.10.16/S0893-6080(03)00115-1. Retrieved 17 November 2013.

[5]. Aäron van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'13), C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.), Vol. 2. Curran Associates Inc., USA, 2643-2651.

[6]. K. Choi, G. Fazekas, and M. Sandler, “Automatic tagging using deep convolutional neural networks,” arXiv preprint arXiv:1606.00298, 2016.

COMPLETE CONTRIBUTARY SOURCE CODE

Content Based Recommender

```
import pandas

import numpy as np

import math



class ContentRecommender:

    def __init__(self,label_map_path = '../../dataset/deep_learning/label_map(boolean_mood).csv',
main_song_label_path = '../../dataset/main_song_labels.csv', main_labels_path =
'../../dataset/main_labels.csv', user_rating_path = '../../dataset/main_user_rating.csv'):

        self.label_map_path = label_map_path

        self.main_song_label_path = main_song_label_path

        self.main_labels_path = main_labels_path

        self.user_rating_path = user_rating_path

        self.song_df = pandas.read_csv(self.main_song_label_path, sep='\t')

        self.TF = None

        self.user_profile = None

        self.rating_df = None

        self.similarityMatrix = None



    def constructLabelVector(self):

        labels_df = pandas.read_csv(self.main_labels_path, sep='\t')
```

```
labels_df['label_count'] = 1


# finding TF

TF = labels_df.groupby(['track_id', 'label'],
as_index=False).count().rename(columns={'label_count': 'label_count_TF'})

label_distinct = labels_df[['label', 'track_id']].drop_duplicates()


# finding DF

DF = label_distinct.groupby('label', as_index=False).count().rename(columns={'track_id':
'label_count_DF'})

a=math.log10(len(np.unique(labels_df['track_id'])))

DF['IDF']=a-np.log10(DF['label_count_DF'])

TF = pandas.merge(TF,DF,on = 'label', how = 'left', sort = False)

TF['TF-IDF']=TF['label_count_TF']*TF['IDF']


# calculating vector

Vect_len = TF[['track_id', 'TF-IDF']]

Vect_len['TF-IDF-Sq'] = Vect_len['TF-IDF']**2

Vect_len = Vect_len.groupby(['track_id'], as_index=False).sum().rename(columns={'TF-
IDF-Sq':'TF-IDF-Sq-Sum'})[['track_id', 'TF-IDF-Sq-Sum']]

Vect_len['vect_len'] = np.sqrt(Vect_len[['TF-IDF-Sq-Sum']].sum(axis=1))
```

```
        # calculating the weight of the song

        TF = pandas.merge(TF, Vect_len, on='track_id', how='left', sort=False)

        TF['label_wt'] = TF['TF-IDF']/TF['vect_len']

        self.TF = TF

        return TF


    def constructUserProfile(self):

        TF = self.TF

        rating_df = pandas.read_csv(self.user_rating_path, sep='\t')

        rating_df = rating_df[rating_df['rating']!=0]

        user_distinct = rating_df['username'].drop_duplicates()

        user_profile = pandas.DataFrame()

        i=1


        for user in user_distinct:

            user_data = rating_df[rating_df['username']==user]

            user_data = pandas.merge(user_data,TF, on='track_id', how='inner')

            user_data_processed = user_data.groupby('label',
as_index=False).sum().rename(columns={'label_wt': 'label_pref'})[['label', 'label_pref']]

            user_data_processed['user'] = user

            user_profile = user_profile.append(user_data_processed, ignore_index=True)
            i+=1


        self.rating_df = rating_df

        self.user_profile = user_profile

        return user_profile


    def constructCosineSimilarity(self):

        rating_df = self.rating_df

        user_profile = self.user_profile

        TF= self.TF

        distinct_users=np.unique(rating_df['username'])

        label_merge_all=pandas.DataFrame()


        i=1

        for user in distinct_users:

            # analizing user data one by one

            print(str(i)+' of '+str(len(distinct_users))+' users')

            user_profile_all= user_profile[user_profile['user']==user]

            distinct_songs = np.unique(TF['track_id'])

            j=1

            for song in distinct_songs:
```

```
            if j%300==0:
                print('song: ', j , 'out of: ', len(distinct_songs) , 'with user: ', i , 'out of: ',
len(distinct_users))


            # analizing song one by one
            TF_song= TF[TF['track_id']==song]
            label_merge = pandas.merge(TF_song,user_profile_all,on = 'label', how = 'left', sort =
False)
            label_merge['label_pref']=label_merge['label_pref'].fillna(0)


            # listing label_value= weight of the label * label profile of the user
            label_merge['label_value']=label_merge['label_wt']*label_merge['label_pref']


            # getting the label weight of the current user-song pair
            label_wt_val=np.sqrt(np.sum(np.square(label_merge['label_wt']), axis=0))


            # getting the label value of the current user-song pair
            label_pref_val=np.sqrt(np.sum(np.square(user_profile_all['label_pref']), axis=0))


            # summing the label_value (rating) of user-song pair
            label_merge_final = label_merge.groupby(['user','track_id']).agg({'label_value':
'sum'}).rename(columns = {'label_value': 'score'}).reset_index()


            # score = score / (label weight * label value)
            label_merge_final['score']=label_merge_final['score']/(label_wt_val*label_pref_val)


            label_merge_all = label_merge_all.append(label_merge_final, ignore_index=True)
            j=j+1
          i=i+1
          label_merge_all=label_merge_all.sort_values(by=['user','score']).reset_index(drop=True)

        self.similarityMatrix = label_merge_all
        return label_merge_all

    def prepareRecommendation(self):
        self.constructLabelVector()
        self.constructUserProfile()
        self.constructCosineSimilarity()

    def recommend(self,username):
        recommendations = 
self.similarityMatrix[self.similarityMatrix['user']==username].sort_values(by='score', 
ascending=False)

        recommendation_detail = pandas.merge(recommendations, self.song_df, on='track_id', 
how='inner')

        return recommendation_detail



if __name__ == '__main__':

    recommender = ContentRecommender()

    recommender.prepareRecommendation()

    print(recommender.recommend('ali').head(10))
```

Song Labeling Model

```
import pandas

import time

import numpy as np

import pdb

from tqdm import tqdm

import librosa



from keras import backend as K

from keras.layers import Input, Dense
from keras.models import Model

from keras.layers import Dense, Dropout, Flatten

from keras.layers.convolutional import Convolution2D

from keras.layers.convolutional import MaxPooling2D, ZeroPadding2D

from keras.layers.normalization import BatchNormalization

from keras.layers.advanced_activations import ELU

from keras.utils.data_utils import get_file

from keras.layers import Input, Dense

from keras.callbacks import ModelCheckpoint


import keras

from sklearn.model_selection import train_test_split


import os.path

import math


class SongLabellingModel():


    def __init__(self, model_path=None, song_preview_dir=None,
label_map_boolean_path=None):

        if model_path:
            self.model_path = model_path
        else:
            self.model_path = 'use6-bestcheckpoint-1501.04- 0.28.hdf5'

        if song_preview_dir:
            self.song_preview_dir = song_preview_dir
        else:
            self.song_preview_dir = '../../dataset/song_preview/'

        if label_map_boolean_path:
            self.label_map_boolean_path = label_map_boolean_path
        else:
            self.label_map_boolean_path =
'../../dataset/deep_learning/label_map(boolean_mood).csv'

        self.model = None

    def buildModelCNN(self):
        K.set_image_dim_ordering('th')

        # Determine proper input shape
```

```
input_shape = (1, 96, 1366)

melgram_input = Input(shape=input_shape)

channel_axis = 1
freq_axis = 2
time_axis = 3

# Input block
x = BatchNormalization(axis=freq_axis, name='bn_0_freq')(melgram_input)

# Conv block 1
x = Convolution2D(64, 3, 3, border_mode='same', name='conv1')(x)
x = BatchNormalization(axis=channel_axis, mode=0, name='bn1')(x)
x = ELU()(x)
x = MaxPooling2D(pool_size=(2, 4), name='pool1')(x)

# Conv block 2
x = Convolution2D(128, 3, 3, border_mode='same', name='conv2')(x)
x = BatchNormalization(axis=channel_axis, mode=0, name='bn2')(x)
x = ELU()(x)
```

```
x = MaxPooling2D(pool_size=(2, 4), name='pool2')(x)


# Conv block 3

x = Convolution2D(128, 3, 3, border_mode='same', name='conv3')(x)

x = BatchNormalization(axis=channel_axis, mode=0, name='bn3')(x)

x = ELU()(x)

x = MaxPooling2D(pool_size=(2, 4), name='pool3')(x)


# Conv block 4

x = Convolution2D(128, 3, 3, border_mode='same', name='conv4')(x)

x = BatchNormalization(axis=channel_axis, mode=0, name='bn4')(x)

x = ELU()(x)

x = MaxPooling2D(pool_size=(3, 5), name='pool4')(x)


# Conv block 5

x = Convolution2D(64, 3, 3, border_mode='same', name='conv5')(x)

x = BatchNormalization(axis=channel_axis, mode=0, name='bn5')(x)

x = ELU()(x)

x = MaxPooling2D(pool_size=(4, 4), name='pool5')(x)


# Output
```

```
    x = Flatten()(x)

    x = Dense(50, activation='sigmoid', name='output')(x)

    # if include_top:

    #     x = Dense(50, activation='sigmoid', name='output')(x)


    # Create model

    model = Model(melgram_input, x)


    return model


# for converting audio file to melgram data

def _compute_melgram(self, audio_path):

    ''' Compute a mel-spectrogram and returns it in a shape of (1,1,96,1366), where

    96 == #mel-bins and 1366 == #time frame


    parameters

    ----------

    audio_path: path for the audio file.

                Any format supported by audioread will work.

    More info: http://librosa.github.io/librosa/generated/librosa.core.load.html#librosa.core.load
```

```
'''


# mel-spectrogram parameters

SR = 12000

N_FFT = 512

N_MELS = 96

HOP_LEN = 256

DURA = 29.12  # to make it 1366 frame..


src, sr = librosa.load(audio_path, sr=SR)  # whole signal

n_sample = src.shape[0]

n_sample_fit = int(DURA*SR)


if n_sample < n_sample_fit:  # if too short

    src = np.hstack((src, np.zeros((int(DURA*SR) - n_sample,))))

elif n_sample > n_sample_fit:  # if too long

    src = src[int((n_sample-n_sample_fit)/2):int((n_sample+n_sample_fit)/2)]

# logam = librosa.logamplitude

logam = librosa.power_to_db

melgram = librosa.feature.melspectrogram

ret = logam(melgram(y=src, sr=SR, hop_length=HOP_LEN,
                        n_fft=N_FFT, n_mels=N_MELS)**2,
                ref=1.0)
        ret = ret[np.newaxis, np.newaxis, :]
        return ret


    def _dataset_to_array(self, df):
        features = np.zeros((0, 1, 96, 1366))


        # df = df.reindex(np.random.permutation(df.index)).head(1000)
        label_matrix = df.copy()
        label_matrix = label_matrix.drop(['track_id', 'song_id', 'title', 'preview_file'],
axis=1).as_matrix()
        for idx,row in tqdm(df.iterrows(), "converting song audio file to features"):
            melgram = self._compute_melgram(self.song_preview_dir+row['preview_file'])
            features = np.concatenate((features, melgram), axis=0)


        if os.path.isfile('song_features-1500.npy'):
            os.remove('song_features-1500.npy')
        if os.path.isfile('song_labels-1500.npy'):
            os.remove('song_labels-1500.npy')
        np.save('song_features-1500.npy', features)
        np.save('song_labels-1500.npy', label_matrix)

        return features, label_matrix


    def _train_in_chunk(self, model, X_train, X_test, Y_train, Y_test, iteration):

        channel = 1

        epochs = 10#50

        batch_size = 10

        verbose = 1


        # normal_checkpoint_name = 'checkpoint-'+str(iteration)+'.{epoch:02d}-{val_acc:
.2f}.hdf5'

        best_checkpoint_name = 'bestcheckpoint-'+str(iteration)+'.{epoch:02d}-{val_acc: .2f}.hdf5'


        # normal_checkpoint = ModelCheckpoint(normal_checkpoint, monitor='val_acc',
verbose=1, save_best_only=True, mode='max')

        best_checkpoint = ModelCheckpoint(best_checkpoint_name, monitor='val_acc', verbose=1,
save_best_only=True, mode='max')

        callback_list = [best_checkpoint]


        model.fit(X_train, Y_train, batch_size=batch_size, epochs=epochs, verbose=verbose,
validation_data=(X_test, Y_test), callbacks=callback_list)

        # model.save(self.model_path)
        path_separated = self.model_path.split('.')
        modelName = path_separated[0]+str(iteration)+'.'+path_separated[1]


        if os.path.isfile(self.model_path):
            os.rename(self.model_path,modelName)
        model.save_weights(self.model_path)
        os.remove('training_status.txt')
        f = open('training_status.txt', 'w')
        f.write('%d' % iteration)
        f.close()


    def train(self, model=None, start=0):
        print(model)
        if model==None:
            model = self.buildModelCNN()


        print(model)


        if os.path.isfile(self.model_path):
            model.load_weights(self.model_path)
```

```
print(model)

model.compile(loss=keras.losses.categorical_crossentropy,

              optimizer=keras.optimizers.Adadelta(),

              metrics=['accuracy'])


if os.path.isfile('randomized.csv'):

   df = pandas.read_csv('randomized.csv', sep='\t')

else:

   df = pandas.read_csv(self.label_map_boolean_path, sep='\t')

   df = df.reindex(np.random.permutation(df.index))

   df.to_csv('randomized.csv', sep='\t', encoding='utf-8', index=False)

df = df[start:]


print(len(df))


chunk_content = 1500 #1000

split_ratio=0.8

chunk_train = chunk_content#int(chunk_content*split_ratio)

random_state=7


# if the numpy file is already there, so no repeated feature sampling again
```

```
        pass_melgram = False


        for i in range(math.ceil(len(df)/chunk_content)):
            if (i*chunk_train)+chunk_content>=len(df):
                part_df = df[i*chunk_train:]
            else:
                part_df = df[i*chunk_train:(i*chunk_train)+chunk_content]


            if pass_melgram:
                pass_melgram = False
                X = np.load('song_features-1500.npy')
                Y = np.load('song_labels-1500.npy')
            else:
                print('Getting features from '+str(i*chunk_train)+'/'+str((i*chunk_train)+len(part_df))+'
    dataset...')
                X, Y = self._dataset_to_array(part_df)
            X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= (1 - split_ratio),
    random_state=random_state, shuffle=True)
            print('X_train shape: '+ str(X_train.shape))
            print('X_test shape: '+ str(X_test.shape))
            print('Y_train shape: '+ str(Y_train.shape))
            print('Y_test shape: '+ str(Y_test.shape))
            del X

            del Y

            print('Training from '+str(i*chunk_content)+'/'+str((i*chunk_content)+len(part_df))+'
dataset...')

            self._train_in_chunk(model, X_train, X_test, Y_train, Y_test, (i*chunk_content)+i)

            del X_train

            del X_test

            del Y_train

            del Y_test

        del df

        self.model = model

        return model


    def getModel(self, train=False, start=0):

        self.model = self.buildModelCNN()

        print(os.path.isfile(self.model_path))

        print(start)

        print(train)

        if train==False:

            print ('a')

            self.model.load_weights(self.model_path)
            return self.model
            # return self.model.load_weights(self.model_path)
        else:
            print ('b')
            # return 'b'
            self.train(self.model, start)
            return self.model

    def predict(self, filepath):
        model = self.model
        feature = self._compute_melgram(filepath)
        result = model.predict(feature)

        return result

if __name__ == '__main__':
    labellingModel = SongLabellingModel()
    model = labellingModel.getModel(True)
```