School of Information Date: April 16, 2023

SI699 Project Report

Music Tagging with Deep Architectures

Mengyan Wu **mengyanw@umich.edu**
Zihui Liu **zihuiliu@umich.edu**
Yuxiao Liu **lyuxiao@umich.edu**

Contents

1 Project statement

The goal of this project is to build up a music tagging system with deep architectures. Specifically, to accurately label music with all the tags, which means the moods or themes of a music in our project, presented in its content. This can be used to structure music collections in a large scale automatically. The digital music market has been growing rapidly and it is becoming increasingly challenging for music lovers and music streaming platforms to organize and categorize large collection of music. Therefore, a music tagging system could offer a solution to this problem by enabling users to efficiently label and sort tracks and improving the music recommendation system in the platforms.

In this project, we experimented with several different deep learning models suggested by state-of-art researches on multi-label music classification, including FCN[1], MusiCNN[2], CNNSA[3], Sample-level CNN[4], CRNN[5] etc. We proposed a MusicBert model that embeds the classification using music title and music content together. We also fine-tuned pretrained models from Hugging Face. We implemented the models based on the same experimental settings and then compared these models' performance. The models were trained on a small portion of the MTG Jamendo Dataset[6] and evaluated using AUC-ROC and corrected tag precision metrics.

We also created an interactive web application to demonstrate our work.

2 Related Work

Several architectures regarding music tagging tasks have been proposed by scholars, most of them being based on deep neural networks.

Content-based music tagging algorithm is proposed based on fully convolutional neural networks (FCN)[1]. The network consists of only convolutional layers and max-pooling layers. Experiments show that the model is effective in improving AUC score for music tagging tasks, but more complex training data is required with the increase in model layers to achieve desired performance.

Sample-level deep convolutional neural networks (DCNN) have been proposed as a powerful way to extract features in music audio for tagging. Choi et al. introduced DCNN model sample-level filters [4]. The model takes raw audio waveforms as input, takes small grains of the waveforms, and extracts features using multiple CNN layers. Experiments show that the model is able to improve the AUC score compared to prior state-of-the arts on both MTAT and MSD dataset.

To integrate local and temporal features from music, Choi et al. proposed a convolutional recurrent neural network(CRNN)[5]. After passing the input through 4-layer CNNs, a 2-layer RNN with gated recurrent unit(GRU) is added to aggregate the temporal features of the input. The model is compared with other CNN-based architectures. Experiments reveal that CRNN is more accurate in predicting music tags, but is also more computationally expensive than its CNN-based counterparts.

The musical domain knowledge justifies the architecture of Musicnn model proposed by Pons et al [2]. The paper tries to explore how different types of deep architectures perform with different size of datasets. The results show that models relying on domain knowledge work better with relatively small datasets, while assumption-free models with waveforms as input work better with large datasets.

Inspired by the success of Transformer model in NLP domain, self-attentions layers are added after the shallow convolutional layers to learn the representation by relating different positions int he sequence [3]. The music tagging model with self-attention is more interpretable and generates competitive results.

With the development of deep learning techniques, research in applying deep architecture models to music tagging task has achieved great successes. Given the different experimental setups, it is hard to make a fair comparison between those models. Based on this, Won et el. [7] provide a comprehensive evaluation of CNN-based automatic music tagging models on three different datasets, which is considered to be a useful reference for further research in this field.

3 Data

3.1 Dataset

The MTG-Jamendo dataset, built with music collected from Jamendo and tags provided by content uploaders[6], is used for our project. The complete dataset contains over 55,000 audio tracks with 195 tags categorized by genre, instruments and mood/theme.

In this project, we only use the mood/theme subset of the whole music data, containing 18486 music tracks, each with one or multiple tags from the 59 unique tags. These tracks have a mean length of 219.4 seconds and a median length of 202.9 seconds.

From the dataset, there are 59 tags, with a distribution shown in Figure 1.

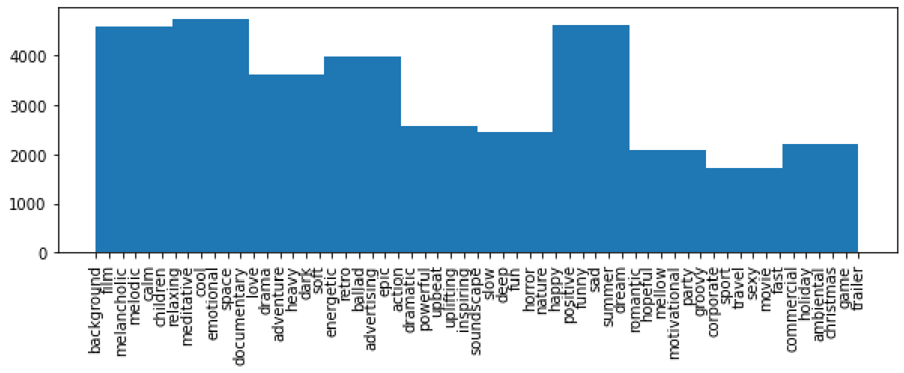


Figure 1: Distribution of the data in 59 mood/theme tags.

The goal of labeling 59 tags increases the complexity in our model. Given the limited resources we have for deep model training, it is quite hard for us to train enough epochs to reach a relatively good result. Besides, we observe that there are tags with similar semantic meanings that can be further grouped into one category. Therefore, to reduce the difficulty of training, we further perform feature dimension reduction to categorize 59 tags into 16 tags as is shown in Table 1. The distribution of new tags is in Figure 2:

Table 1: Mapping 59 tags into 16 tags.

New tag	Mapping
film	film, movie, trailer, documentary
uplifting/euphoric	energetic, upbeat, uplifting, powerful, summer
relaxing/soothing	relaxing, calm, meditative, slow, soft, mellow, dream
somber	emotional, sad, melancholic, deep, ballad
corporate	advertising, commercial, corporate
festive	party, sport, holiday, christmas
thrilling	horror, action, adventure, fast
rhythmic	sexy, melodic, groovy, retro
inspiring	inspiring, motivational, hopeful, travel
joyful	children, fun, funny, game
happy	happy, positive
love	love, romantic
drama	drama, dramatic
dark/heavy	heavy, dark
grand/powerful	epic, space
ambiental	soundscape, background, cool, ambiental, nature

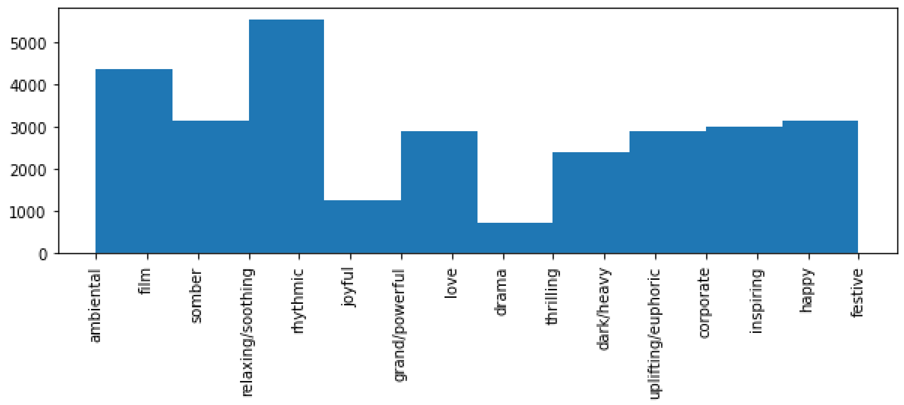


Figure 2: Distribution of the data in 16 mood/theme tags.

3.2 Audio preprocessing

In order for the digital system to understand audio and to transform the audio into numeric data to fit into our model, we need to sample and extract features from music signals. It is important for those data to capture all the information in good quality. In the following section, we will elaborate on different strategies of audio preprocessing and explore how these methods will affect the result of music tagging.

Waveform A waveform is the time-domain representation of an audio, which displays the changes in a audio signal's amplitude over time. The sampling rate is the average number of samples taken per second from the waveform, which a continuous signal, to create a discrete signal. For consistency matter, we resample the audio data as 16kHz. The waveform generated from one of the audio in our dataset with sampling rate 16kHz can be seen in Figure 3. Considering the size of the data and our limited computational resources, we only sample 15 seconds of the audio. Audio with duration smaller than 15s will be padded using the mean value. The `librosa` package in Python is used.

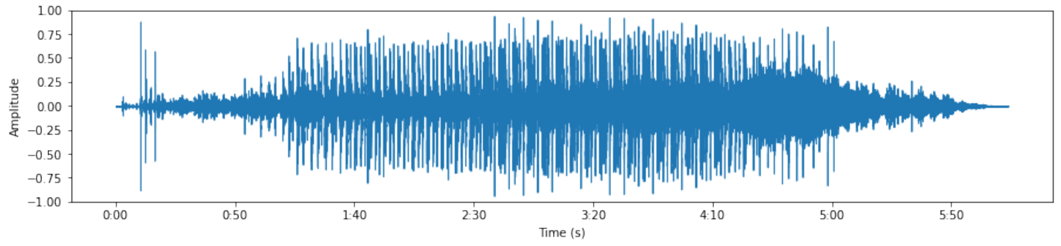


Figure 3: Waveform of an audio with sampling rate 16kHz.

Mel Spectrogram The amplitudes of the music changes over time are not very informative when trying to predict the mood/theme in a music. We are also interested in frequency-domain information. Fast Fourier Transform (FFT) is used to compute the discrete Fourier transform (DFT), where it converts a discrete signal from time-domain to frequency-domain. In order to not lose track of the time information, the audio signal can be broken into smaller frames and go through FFT. In this way, we can get the frequency for each window and the window number represent the time. In this way, we obtained the other representation of audio called spectrogram. A spectrogram representation S consists of 2 dimensions, where one dimension represents time while one represents frequency. The entry value $S[x][y]$ represents the amplitude of y frequency at time x. The algorithm is called short-time Fourier transform (STFT).

Studies have shown that humans do not perceive frequencies on a linear scale. Human tend to detect differences in lower frequencies easier compared to higher frequencies. In order to mimic our perception of pitch, Stevens et al. proposed the mel scale [8] where equal distances in the scale sounded equally distant to the listener. We can get the mel spectrogram by mapping the frequency onto the mel scale. An example plot of mel spectrogram can be seen in Figure 4. The brighter the color, the stronger the amplitude.

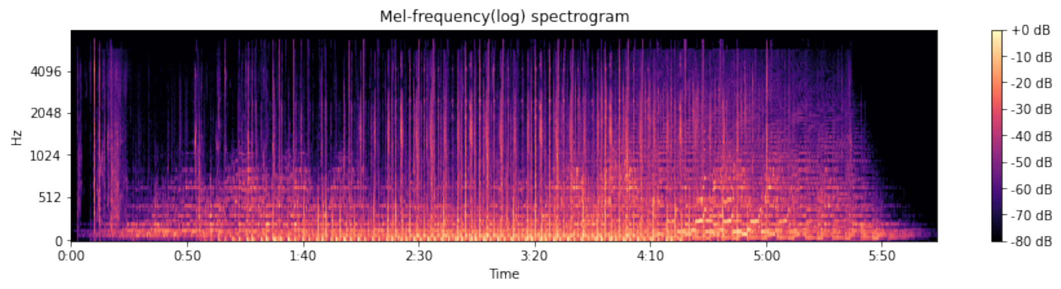


Figure 4: Mel-spectrogram of an audio.

Besides, there is also a logarithmic compression of amplitudes, namely, decibel scaling. The scale is motivated by the logarithmic relationship between the human perception of loudness and the physical energy of sound [9], which is considered as a standard preprocessing method in music information retrieval.

In this project, we set the size of FFT to be 512, number if mel filterbanks to be 128, minimum and maximum frequency to be 0 and 8000 separately, and the length of hop between STFT windows to be 40. The `torchaudio` package in Python is used to perform this step.

4 Methodology

A music tagging model [1] can be broken into preprocessing, front-end, and back-end modules, as can be seen in Figure 5.

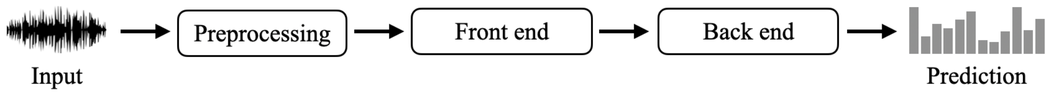


Figure 5: Framework of music tagging model.

In this project, we adopted different deep architecture models proposed by previous state-of-art studies to perform the multi-label classification task on the dataset based on the same experimental setups [7]. We also fine-tuned the pretrained model from huggingface. The goal of our model is to recognize the mood/theme in a music.

4.1 Baselines

We proposed two baseline models for comparison, adopted from [2].

Popularity baseline The first baseline we applied is the naive popularity baseline. The popularity baseline model always predicts the most frequent tag among all the tracks in the training dataset.

VGG-ish baseline The VGG model baseline consists of five 2D convolutional layers followed by a dense connection.

4.2 State-of-art models

The implementation of models in this section are adopted from the github repo [3].

4.2.1 FCN

Mel spectrograms are taken as the input for the front-end. As one of the earilest approaches, the architecture only contain a front-end architecture.The model[1] is composed of several convolutional layers, each followed by a max pooling layer. In our case, 5 is chosen as the number of convolutional layers, since we have a middle-scale data size and therefore requires a larger layer size to match up with it. For each convolutional layer, we choose a kernel size of either 3 or 4. This ensures that essential features are scanned within each block while retaining a reasonable computational speed. The max pooling layer reduces the feature map size while capturing the main information over an area.

[1] https://music-classification.github.io/tutorial/landing-page.html
[2] https://multimediaeval.github.io/2021-Emotion-and-Theme-Recognition-in-Music-Task/
[3] https://github.com/minzwon/sota-music-tagging-models

4.2.2 Sample-level CNN

Sample-level CNN [4] aims to learn representations from very small grains of waveforms (e.g. 2 or 3 samples) instead of using typical spectrogram representations. It then consists of several 1D convolutional filters where each filter is consitituted by a 1D convulution, a 1D batch normalization, a ReLu non-linearity, and max-pooling for sub-sampling. Given that there is only 1D convolutional layers, there is no clear boundary of front-end and back-end.

4.2.3 CRNN

Similar to FCN, mel spectrograms from binary clips are taken as the input. Four 2d convolutional layers serve as the front-end structure to extract local features. A 2-layer RNN with gated recurrent unit(GRU) serves as the back-end to extract temporal features. GRU selectively update the hidden states at each time step, which is especially useful when handling long sequences and thus preventing vanishing gradient problem.

4.2.4 Musicnn

The model [2] consists of 2 parts, a CNN spectrogram front-end and a CNN back-end.

1. The spectrogram front-end takes log-mel spectrograms as input. Inputs go through normalization layer to have zero-mean and unit-var. The design of the first convolutional layer takes the domain knowledge into consideration, which consists of vertical and horizontal filters.
   (a) Vertical filters are designed to capture pitch-invariant timbral features, i.e., capture the timbre ("color" or the "quality" of a sound) regardless of the pitch in which it is played.
      - A convolutional layer comes first to extract features.
      - The max-pooling layer is applied after convolutional layer to map features across their vertical axis ensures pitch invariance.

   (b) Horizontal filters are meant to learn temporal features (describe the relatively long-term dynamics of a music signal over time) with long filter shapes.
      - A mean-pooling layer is applied to the frequency-axis of the spectrogram.
      - A convolutional layer receives the average-pooled spectrograms as inputs.
2. The extracted timbral and temporal features from the frond-end are concatenated and are fed as input the the back-end model. The back-end consisting of 3 CNN layers, 2 pooling layers and a dense layer summarizes the features to predict the tags.

4.2.5 CNNSA

Self-attention in Transformer has achieved state-of-the-art result in the field of NLP. Music composes its semantics based on the relations between components in sparse positions. To learns the representation by relating those relations, a self-attention layer is applied after the shallow convolutional layers. The structure is composed of 2 parts, a CNN spectrogram front-end and a self-attention back-end. The front end is the same as the one used in Musicnn model. While for the back-end, instead of using CNN or RNN, the paper [3] proposed using a self-attention layer in the back end to capture more structural information such rhythmic patterns and melodic contours.

4.3 MusicBert Tagger

The title of a music often comes together with a music track, which also indicates information regarding the mood/theme. To utilize this information, we extract the titles of each track from the meta data provided by MTG-Jamendo dataset and apply NLP techniques to them. The detailed designs are described as follows and also in Figure 6:

1. BERT [10] is a famous NLP model that is able to achieve state-of-the-art results in a variety of NLP tasks by applying Transformer to its architecture. We first use a pretrained `BertTokenizer` from Huggingface [4] to get the word embedding of the title. `BertTokenizer` generates 2 set of tensors that are passed as input to our models, `input_ids` being the indices of tokens in the vocabulary and `attention_mask` indicating to the model which tokens should be attended to and which should not.

2. We embed the text tagging model together with the previous music tagging model. Namely, instead of creating a new model and training separately, we add Bert model layers into the music tagging model to form as a single model. The output of Bert model will be mapped through multiple dense layers into a vector v_1 of # tags dimension. v_1 will be concatenated with v_2, which is the output vector of music tagging model. The result vector v_3 has a dimension of $2 \times$ # tags. v_3 will then go through a dense and a sigmoid layer to get the result logits.

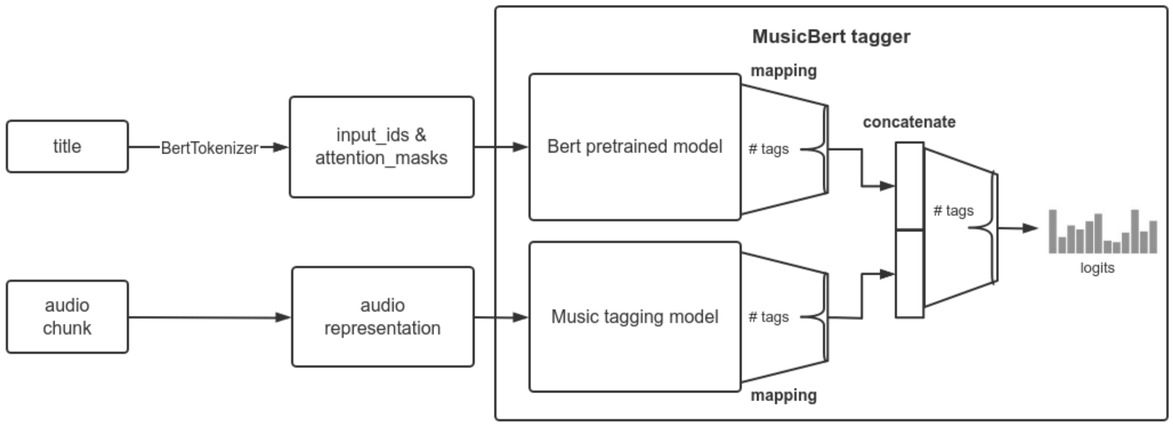


Figure 6: Our proposed MusicBert Tagger diagram.

4.4 Fine-tune Wav2vec2

Music and speech signals are similar in the ways that they both are time-varying signals and can be applied to similar signal processing techniques such as Fourier Transform. Both music and speech signals can be learned using deep learning models. There are lots of competitive research results in the field of speech audio representation and recognition. Wav2vec2 [11] is a state-of-the-art speech recognition model that was introduced by Facebook AI Research (FAIR) in 2020. Its way of training allows users to load a pre-train model and be fine-tuned on a particular dataset for a downstream task. In this project, we explore the feasibility of adapt the pre-trained speech models to music tasks using the pretrained wav2vec2 model [5]. The process is described as follows:

[4]https://huggingface.co/bert-base-uncased/tree/main
[5]https://huggingface.co/docs/transformers/model_doc/wav2vec2

1. Load the pre-trained `Wav2Vec2FeatureExtractor` and feed the raw waveform into it to extract the features from audio.

2. Load the pre-trained wav2vec2 model `Wav2Vec2Model` from `"facebook/wav2vec2-base-960h"`. The model takes the output of feature extractor as input.

3. Add several dense layers, dropout layers, and also a sigmoid layer to get the probabilities over different tags. Train the models on our dataset.

5 Result & Analysis

5.1 Evaluation Metrics

Two metrics, multi-label Compute Area Under the Receiver Operating Characteristic Curve(AUC-ROC) and correct tag percentage are employed to evaluate the performance of different models.

- The AUC-ROC for a single label is the area under the ROC curve(True Postivie Rate v.s. False Positive Rate curve). To calculate the AUC-ROC score for multiple labels, the AUC-ROC for each label is calculated separately and then the average is taken. The value measures the model's performance in distingushing between postive and negative classes.

- The correct tag percentage, different from the overall accuracy, evaluates the percentage of correctly predicted tags over the sum of ground truth labels. It is a decisive metric that is designed by us to correspond to our goal. The goal of our project can be formulated mathematically as follows:

  – For each data entry, the label vector v_l is a combination of 0s and 1s, e.g. $[0, 1, \ldots, 0, 0, 1, 1, 0]$. We would like our model to output a vector of probability over each tag v_o where for each tag index i, if $v_l[i] = 1$, $v_o[i]$ should be as close to 1 as possible, while if $v_l[i] = 0$, $v_o[i]$ should be as close to 0 as possible.

  In this way, the correct tag percentage can be calculated as:

  – Let k denotes the number of 1 in v_l, which is the number of tags this music presents. The top k entries in v_o are labeled as 1, while other entries are labeled as 0, formulated a new vector v_o. The correct tag percentage is equal to the number of matched 1 in v_l and v_c divided by k.

5.2 Evaluation Results

Based on previous discussion of music tagging methodologies, 7 models are implemented and their performance on the MTG-Jamendo dataset[6] are evaluated together with our benchmark models. We divide our dataset, which composed of in total 18486 data, into train set (%80) and validation set (%20). The validation set is used for model selection. For all models, the learning rate is 0.0001 (except for baseline 1 where no learning rate is needed) and the batch size is set to 4. In order to keep consistent input size and also considering the limited resources we have, all music data are clipped to 15 seconds for this project.

Depending on the input requirements of different models, we experimented over different data transformers and results on validation set for individual models are compared. Only the best prediction results for each model are shown in this part, and a complete version of the experiment results is in Appendix. Note that it can be seen in the following tables that the number of epochs of result is not consistent for different models. This is due to the reason that we only record the best correct tags precision over the epochs. Some models do not exhibit convergence with the increase of the epoch.

5.2.1 59 tags

First the performance over all 59 mood/theme tags are evaluated. Table 2 shows the best prediction results for each model on the test set.

Table 2: Performance of different models for 59 tags

model	**data transformer**	**# epoch**	**tag pren**	**AUC-ROC**	**loss**
baseline1 (most frequent)	none	none	0.0340	0.5000	none
baseline2 (VGG-ish)	logmel [6]	5	0.1296	0.6036	0.1266
SampleCNN	waveform	10	0.1290	0.5679	0.1402
Musicnn	logmel	14	0.1324	0.5799	0.1342
MusicBert	logmel	40	0.1204	0.6129	0.1250
FCN	logmel	20	0.1531	0.6568	0.1239
CRNN	**logmel**	**20**	**0.1592**	**0.6784**	**0.1224**
CNNSA	logmel	2	0.1330	0.5820	0.1277

5.2.2 16 tags

Then we evaluate the models' performance over 16 tags mapped from the 59 original tags. Table 3 shows the best prediction results for each model on the test set.

Table 3: Performance of different models for 16 tags

model	**data transformer**	**# epoch**	**tag pre**	**AUC-ROC**	**loss**
baseline1 (most frequent)	none	none	0.0879	0.5000	none
baseline2 (VGG-ish)	logmel	1	0.2623	0.5661	0.3105
SampleCNN	waveform	29	0.2684	0.5556	0.5011
Musicnn	**logmel**	**40**	**0.8714**	**0.9715**	**0.1020**
MusicBert	logmel	40	0.6928	0.8990	0.2706
wav2vec	wav2vec	5	0.2446	0.5327	0.3298
FCN	melspec	20	0.3196	0.6765	0.3003
CRNN	logmel	20	0.3592	0.7094	0.2868
CNNSA	logmel	8	0.2905	0.6673	0.3269

[6] Performed decibel scaling on mel spectrogram

5.3 Analysis

When 59 tags are taken into consideration, performance does not differ much among different models. Some models even do not perform as good as the VGG-ish baseline model. However, tag precision and AUC-ROC score converge relatively quickly with the increase in epochs. This might due to the reason that the relatively high dimension in feature space increases the complexity of the problem, without training on enough epochs and enough data, it is hard for the model to learn the music and converge to a good and stable result. Another reason is that there are some tags with very close semantic meaning, for example, ”drama” and ”dramatic”, ”fun” and ”funny”, making it difficult for the model to distinguish between them. It is superising to see that the CRNN model with logmel data transformer performs better than the others. One possible explanation might be RNN's better capability of capturing temporal features in this problem. However, CRNN also displays unstability of results between epochs.

Compared with the 59-tags problem, the performance of models improved greatly with the decrease in tag categories. When tags are more roughly mapped, Musicnn significantly outperforms the other models in all aspects. Musicnn might benefit from its complex front-end architecture: vertical long filters are used to capture timbral characters, while horizontal filters capture temporal rhythmic patterns. Besides, although the design choices of Musicnn restrict it from learning more information on larger datasets, it takes advantage of domain knowledge when the dataset is small, which can explain why Musicnn performs better in our project.

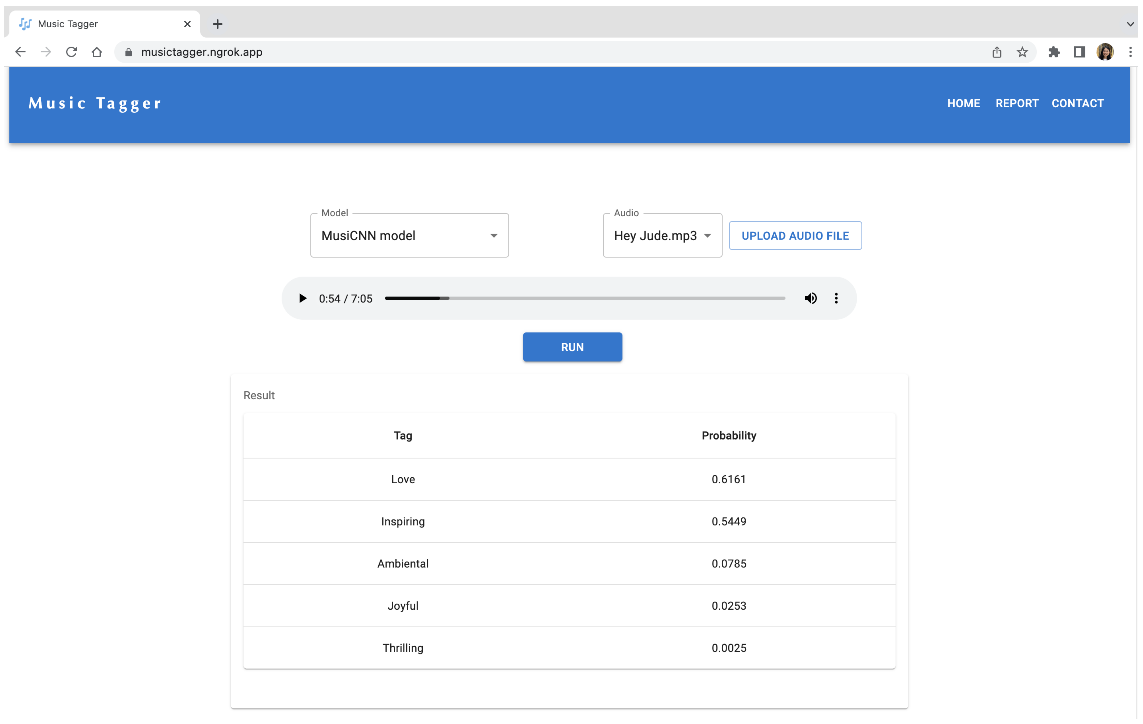
MusicBert, on the other hand, does not get a better result than Musicnn with more information added. An explanation could be that when loading the pre-trained Bert model into MusicBert, we do not freeze the parameters, which means that the parameters is being updated in each learning process, but the title of most songs are too short to generate semantic meanings that are useful for classification

CNN with self-attention layers fail to outperform the CRNN model. According to visualizations from Won's work [3], attention layers always pay attention to the more informative part of the spectrogram. For example, the model pays attention to the loud part of the audio although the spectrogram was classified as ”quiet”. This feature might confuse the model and make it predict opposite tags. To validate this, in future work, we could take a look at the precision of some tags.

Under most circumstances, models with logmel data transformer yield better results than ones with melspec transformer. One possible explanation might be that human auditory system perceives loudness in a logarithmic way, so that a logarithmic filter better corresponds to human perception[12].

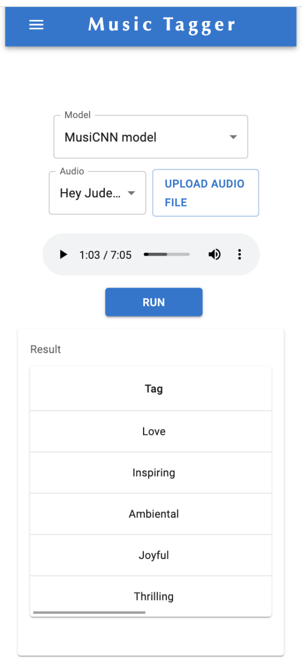
Besides, it can be noticed that the value of correct tag precision is positively correlated with the AUC-ROC, which indicates that the design logic of correct tag precision is reasonable.

6 Website



Tag	Probability
Love	0.6161
Inspiring	0.5449
Ambiental	0.0785
Joyful	0.0253
Thrilling	0.0025

Figure 7: Desktop view of the website.



Tag
Love
Inspiring
Ambiental
Joyful
Thrilling

Figure 8: Mobile view of the website.

We built a demo website to demonstrate our work. Users could run our models with the provided or uploaded music tracks on the website.

We provide 3 implementation methods. The code is available in our team's Github repository. [7]

6.1 React & Flask web application

The first method is to use React and Flask to build the website. The front UI was implemented with React (JavaScript) and Material UI. The backend server was implemented with Flask (Python).

On the website, users could select a model from MusiCNN, FCN, Sample-level CNN, CRNN, and CNNSA, and choose the track from provided examples or upload local mp3 files. After users click the 'Run' button in the UI, the front end would send a request to the server. The flask server will then execute the prediction function with the model chosen by the user. After execution, the server would send back the result to the front end. The website will then display the top 5 most possible tags along with the possibilities.

We also built a page to embed the slides and project report in the web application.

6.2 Static React & ONNX models web application

The second method used React only for the website. This allows the website to be a completely static application that does not requires a backend server to run on.

But this implementation does have limitations. To predict tags with models purely on the front end, we need to convert the PyTorch model into a JavaScript-compatible format, the ONNX (Open Neural Network Exchange) format. However, this conversion is limited to simple models with very few layers and it is hard to fit in models with deeper structures. We ended up only being able to provide two model choices that have lower performance than the CNN variation models.

6.3 Streamlit

The third method used Streamlit to build the app. Streamlit is an open-source app framework for data science teams. It allows the developer to create web app with pure python in a fast yet neat way.

7 Discussion

7.1 Dataset

Due to the nature of audio files, the dataset is much larger comparing to other machine learning tasks when the numbers of data entries are the same. The huge training set leads to problems with the model training part. We only have limited GPU resources and storage to load large enough data perform the training, while we still want to achieve good performance.

The first thing we do is to trim the audio file so that we are only working on a shorter clip instead if the entire music file. This helps us to speed up the training process and limit the file size so that more tracks could be trained. Before trimming audio files, we tried to train on only 3 folders of the whole dataset(1200 tracks), each with the entire music length included. However, the results of this initial trail turns out to be very unstable between epoches. One possible reason may be the uneven distribution of tags in a small dataset. Therefore, we enlarged the dataset size to include more samples

[7] https://github.com/Jenniferlyx/si699-music-tagging

while trim the audio to be only 15 seconds and thus get a more stable results. This can be seen as a tradeoff because training with a small chunk of data is noisier since it is possible that the trimmed does not contain all of mood presents in an audio, but we can expect a much larger of data for training.

To improve the result we also limited the number of the tags to 16 by categorizing the original 59 tags. Tags with similar semantic meanings are grouped together. This eliminate redundant tags and reduce the complexity of the problem a lot.

7.2 Model Structure

A unified input data dimension is maintained across all models. However, different models could expect a variation in performance when the initial input size changes. Among the models that we deployed, many of them do not have a input specification stated in their original paper. Even when the input data is specified, these conditions may only apply to the datasets upon which the models are trained.

The same situation applied to the detailed architecture of models.Many models do no offer information about suggested kernel size, padding size and other parameters in convolutional layers, and we also made adjustments to these architectures based on our data shape. However, these aspects are worth experimenting on in the future, especially for understanding their effects on extracting audio features.

7.3 Website

We tried to run our music tagger using track URLs from music streaming services like Spotify. However, after examining the APIs provided by the companies, we found that due to copyright limitations, accessing most of the songs requires user login and membership verification. This will lead to complex redirects and authentications (users will need to leave the current page and log in to their music streaming account in order to proceed), which are not flexible enough for a light demo website.

Due to the problem stated above, we chose to provide more example music choices instead. Users are also welcomed to upload local audio files to the website.

8 Conclusion

Deep structure-based music taggers have shown great potential in automatically tagging and organizing large music datasets. Deep learning models can capture complex patterns and features in the music signal and significantly improve the efficiency and accuracy of music tagging.

In this project, we built and compared several deep structure-based music tagging models, including FCN, MusiCNN, CNNSA, Sample-level CNN, and CRNN. We also proposed a MusicBert model that generates music tags based on both the track title and the music signal. We trained and evaluated the models under the same experimental settings. Among all these models, MusiCNN has the highest tag precision and AUC-ROC score.

The relatively large size of the audio file, and the complexity of music moods and genres make it challenging for finding the best practice among these music tagging models. We proposed multiple methods to speed up the training process and to improve the performance of the models, which might be useful for future studies in related fields.

We also built an interactive web application where people could choose from the models, and then see the prediction result of that model on selected or uploaded music tracks in a few clicks. People with non-data science backgrounds could still use this website to tag the music tracks with ease.

There is still potential for further improvement if the models were trained with a larger dataset and more computing resources. Future studies may result in more accurate and robust models that can handle a wider range of music tags. Nonetheless, our study of different deep structured based music tagging models have achieved promising results and may serve as a useful reference when choosing the models.

9 Statement of work

Yuxiao and Zihui processed the dataset, implemented and trained the models.

Mengyan built the website and evaluated the results.

All team members shared the work of contributing to github repository, writing the report and preparing presentations.

References

[1] Keunwoo Choi, George Fazekas, and Mark Sandler. Automatic tagging using deep convolutional neural networks. *arXiv preprint arXiv:1606.00298*, 2016.

[2] Jordi Pons, Oriol Nieto, Matthew Prockup, Erik Schmidt, Andreas Ehmann, and Xavier Serra. End-to-end learning for music audio tagging at scale. *arXiv preprint arXiv:1711.02520*, 2017.

[3] Minz Won, Sanghyuk Chun, and Xavier Serra. Toward interpretable music tagging with self-attention. *arXiv preprint arXiv:1906.04972*, 2019.

[4] Jongpil Lee, Jiyoung Park, Keunhyoung Luke Kim, and Juhan Nam. Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms. *arXiv preprint arXiv:1703.01789*, 2017.

[5] Keunwoo Choi, György Fazekas, Mark Sandler, and Kyunghyun Cho. Convolutional recurrent neural networks for music classification. In *2017 IEEE International conference on acoustics, speech and signal processing (ICASSP)*, pages 2392–2396. IEEE, 2017.

[6] Dmitry Bogdanov, Minz Won, Philip Tovstogan, Alastair Porter, and Xavier Serra. The mtg-jamendo dataset for automatic music tagging. In *Machine Learning for Music Discovery Workshop, International Conference on Machine Learning (ICML 2019)*, Long Beach, CA, United States, 2019.

[7] Minz Won, Andres Ferraro, Dmitry Bogdanov, and Xavier Serra. Evaluation of cnn-based automatic music tagging models. In *Proc. of 17th Sound and Music Computing*, 2020.

[8] S. S. Stevens, J. Volkmann, and E. B. Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937.

[9] Brian C. J. Moore. *An Introduction to the Psychology of Hearing.* Academic Press, 2003.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

[11] Alexei Baevski, Heng-Jui Zhou, Abdelrahman Mohamed, and Michael Auli. Wav2vec 2.0: A framework for self-supervised learning of speech representations. In *International Conference on Learning Representations (ICLR)*, 2020.

[12] Jash Mehta, Deep Gandhi, Govind Thakur, and Pratik Kanani. Music genre classification using transfer learning on log-based mel spectrogram. In *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, pages 1101–1107. IEEE, 2021.

Appendix A: Performance of different models for 59 tags

model	data transformer	# epoch	tag pre	AUC-ROC	loss
baseline1 (most frequent)	none	none	0.0340	0.5000	none
baseline2 (VGG-ish)	logmel	5	0.1296	0.6036	0.1266
SampleCNN	waveform	10	0.1290	0.5679	0.1402
Musicnn	logmel	14	0.1324	0.5799	0.1342
MusicBert	logmel	40	0.1204	0.6129	0.1250
Musicnn	melspec	10	0.1304	0.5793	0.1311
ShortChunkCNN_Res	logmel	2	0.1221	0.5608	0.1442
ShortChunkCNN_Res	melspec	4	0.1278	0.5659	0.1780
fcn	melspec	20	0.1401	0.6153	0.1264
fcn	logmel	20	0.1531	0.6568	0.1239
crnn	melspec	20	0.1471	0.643	0.1251
crnn	**logmel**	**20**	**0.1592**	**0.6784**	**0.1224**
cnnsa	logmel	2	0.1330	0.5820	0.1277
cnnsa	melspec	5	0.1258	0.6008	0.1250

Appendix B: Performance of different models for 16 tags

model	data transformer	# epoch	tag pre	AUC-ROC	loss
baseline1 (most frequent)	none	none	0.0879	0.5000	none
baseline2 (VGG-ish)	logmel	1	0.2623	0.5661	0.3105
SampleCNN	waveform	29	0.2684	0.5556	0.5011
Musicnn	**logmel**	**40**	**0.8714**	**0.9715**	**0.102**
MusicBert	logmel	40	0.6928	0.8990	0.2706
Musicnn	melspec	19	0.2994	0.6413	0.3043
ShortChunkCNN_Res	logmel	10	0.2614	0.5639	0.3159
ShortChunkCNN_Res	melspec	11	0.2604	0.5540	0.4325
fcn	melspec	20	0.3196	0.6765	0.3003
fcn	logmel	20	0.2819	0.6016	0.3041
crnn	melspec	20	0.3142	0.6550	0.3074
crnn	logmel	20	0.3592	0.7094	0.2868
cnnsa	logmel	8	0.2905	0.6673	0.3269
cnnsa	melspec	7	0.2636	0.5691	0.3106